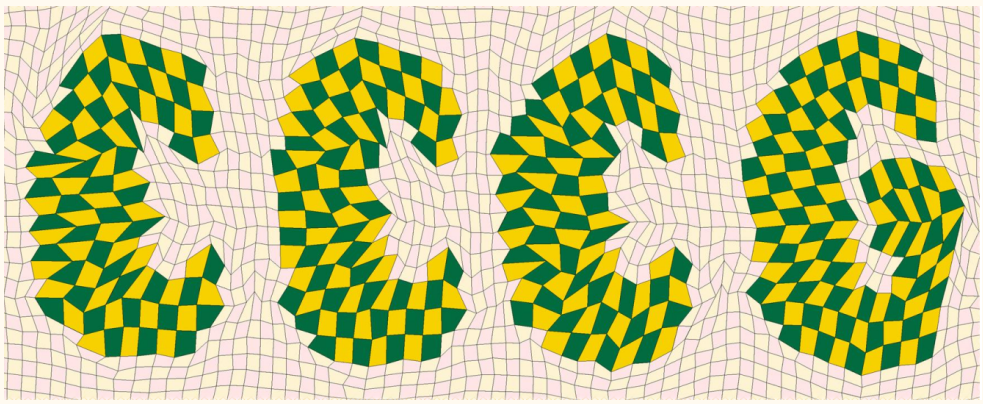


Proceedings of the 32nd Canadian Conference on Computational Geometry (CCCG 2020)

August 5-7, 2020
University of Saskatchewan
Saskatoon, Saskatchewan
Canada

Logo designed by the Visualization, Geometry, and Algorithms Lab at the University of Saskatchewan

Preface

This volume contains the proceedings of the 32nd Canadian Conference on Computational Geometry (CCCG 2020), which was organized on August 5-7, 2020 at University of Saskatchewan, Saskatoon, Saskatchewan and hosted online. These papers are also available electronically at CCCG 2020 website: http://vga.usask.ca/cccg2020/.

We are grateful to the Program Committee, and external reviewers, who thoroughly examined all submissions and provided excellent feedback. Each submission was reviewed by an average of three program committee members. Out of 68 papers submitted, the committee decided to accept 47 papers. We thank the authors of all submitted papers, all those who have registered, and in particular the invited speakers Erik Demaine (Paul Erdős Memorial Lecture), Jeff Erickson (Ferran Hurtado Memorial Lecture), and Yusu Wang (Godfried Toussaint Memorial Lecture). Many thanks go out to the local arrangements committee for their organizational assistance.

We gratefully acknowledge financial support from the Pacific Institute for the Mathematical Sciences (PIMS), Elsevier, the Fields Institute for Research in Mathematical Sciences, and University of Saskatchewan.

Mark Keil
Debajyoti Mondal
CCCG 2020 Program Committee Co-Chairs

Sponsored by

Invited Speakers

Erik Demaine	Massachusetts Institute of Technology, USA
Jeff Erickson	University of Illinois at Urbana-Champaign, USA
Yusu Wang	Ohio State University, USA

Program Committee

Esther M. Arkin	Stony Brook University, USA
Therese Biedl	University of Waterloo, Canada
Ahmad Biniaz	University of Windsor, Canada
Sergio Cabello	University of Ljubljana, Slovenia
Jean Cardinal	University Libre de Bruxelles, Belgium
Erin W. Chambers	Saint Louis University, USA
Guilherme D. da Fonseca	Aix-Marseille University, France
Mirela Damian	Villanova University, USA
Vida Dujmović	University of Ottawa, Canada
Stephane Durocher	University of Manitoba, Canada
William Evans	University of British Columbia, Canada
Mark Keil (co-chair)	University of Saskatchewan, Canada
Irina Kostitsyna	Eindhoven University of Technology, Netherlands
Anna Lubiw	University of Waterloo, Canada
Victor Milenkovic	University of Miami, USA
Tillmann Miltzow	Utrecht University, Netherlands
Debajyoti Mondal (co-chair)	University of Saskatchewan, Canada
Pat Morin	Carleton University, Canada
Amir Nayyeri	Oregon State University, USA
Joseph O'Rourke	Smith College, USA
Rodrigo I. Silveira	University Politècnica de Catalunya, Spain
Csaba D. Tóth	California State University Northridge, USA
Ryuhei Uehara	JAIST, Japan
Norbert Zeh	Dalhousie University, Canada

Additional Reviewers

Mikkel Abrahamsen, Hugo Akitaya, Andrei Asinowski, Aritra Banik, Aaron Becker, Mitchell Black, Kaustav Bose, Kyle Fox, Yan Gerard, Adam Hesterberg, Michael Hoffmann, David Kirkpatrick, Marc Van Kreveld, Jason S. Ku, Hung Le, Maarten Löffler, William Maxwell, Nabil Mustafa, Anurag Murty Naredla, Parthiban Natarajan, Rahnuma Islam Nishat, Benjamin Raichel, André van Renssen, Toshiki Saitoh, Noushin Saeedi, Carlos Seara, Frank Staals, Giovanni Viglietta, Kunihiro Wasa, Hamid Zarrabi-Zadeh

Local Organizers

Mohammad Rakib Hasan
Mark Keil (Co-chair)
Ehsan Moradi
Debajyoti Mondal (Co-chair)
Sakib Mostafa

(All at University of Saskatchewan)

Table of Contents

Wednesday, August 5

Paul Erdős Memorial Lecture

Tribute to Godfried Toussaint 1
Erik Demaine

Session 1A

Minimum Ply Covering of Points with Convex Shapes 2
Ahmad Biniaz and Zhikai Lin

Convex Hull Complexity of Uncertain Points 6
Hongyao Huang and Benjamin Raichel

Sparse Convex Hull Coverage 15
Georgiy Klimenko, Benjamin Raichel and Gregory Van Buskirk

Fair Covering of Points by Balls 26
Daniel Lokshtanov, Chinmay Sonar, Subhash Suri and Jie Xue

Covering Points with Pairs of Concentric Disks 33
Anil Maheshwari, Saeed Mehrabi, Sasanka Roy and Michiel Smid

Hardness of Approximation for Red-Blue Covering 39
Sima Hajiaghaei Shanjani

Session 1B

Relocating Units in Robot Swarms with Uniform Control Signals is PSPACE-Complete 49
David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller and Tim Wylie

Building Patterned Shapes in Robot Swarms with Uniform Control Signals 56
David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller and Tim Wylie

New Results in Sona Drawing: Hardness and TSP Separation 63
Man-Kwun Chiu, Erik D. Demaine, Yevhenii Diomidov, David Eppstein, Robert A. Hearn, Adam Hesterberg, Matias Korman, Irene Parada and Mikhail Rudoy

Minimizing The Maximum Distance Traveled To Form Patterns With Systems of Mobile Robots 73
Jared Coleman, Evangelos Kranakis, Oscar Morales-Ponce, Jaroslav Opatrny, Jorge Urrutia and Birgit Vogtenhuber

Path Planning in a Weighted Planar Subdivision Under the Manhattan Metric 80
Mansoor Davoodi, Hosein Enamzadeh and Ashkan Safari

Scheduling Three Trains is NP-Complete 87
Christian Scheffer

Thursday, August 6

Ferran Hurtado Memorial Lecture

Chasing Puppies 94
Jeff Erickson

Session 2A

Folding Small Polyominoes into a Unit Cube 95
Kingston Yao Czajkowski, Erik D. Demaine, Martin L. Demaine, Kim Eppling, Robby Kraft, Klara Mundilova and Levi Smith

Some Polycubes Have No Edge Zipper Unfolding 101
Erik D. Demaine, Martin L. Demaine, David Eppstein and Joseph O'Rourke

Acutely Triangulated, Stacked, and Very Ununfoldable Polyhedra 106
Erik D. Demaine, Martin L. Demaine and David Eppstein

Nets of higher-dimensional cubes 114
Kristin DeSplinter, Satyan Devadoss, Jordan Readyhough and Bryce Wimberly

Efficient Folding Algorithms for Regular Polyhedra 121
Tonan Kamata, Akira Kadoguchi, Takashi Horiyama and Ryuhei Uehara

Vertex-Transplants on a Convex Polyhedron 128
Joseph O'Rourke

Session 2B

Fitting a Graph to One-Dimensional Data 134
Siu-Wing Cheng, Otfried Cheong and Taegyoung Lee

Planar Emulators for Monge Matrices 141
Hsien-Chih Chang and Tim Ophelders

Simultaneous Visibility Representations of Undirected Pairs of Graphs 148
Ben Chugg, William S. Evans and Kelvin Wong

Finding a Maximum Clique in a Grounded 1-Bend String Graph 160
Mark Keil, Debajyoti Mondal and Ehsan Moradi

Testing Balanced Splitting Cycles in Complete Triangulations 167
Vincent Despré, Michaël Rao and Stéphan Thomassé

A Linear-Time Algorithm for Discrete Radius Optimally Augmenting Paths in a Metric Space 174
Haitao Wang and Yiming Zhao

Thursday, August 7

Godfried Toussaint Memorial Lecture

Topological and geometric methods for graph analysis . . . 181
Yusu Wang

Session 3A

Computing the Caratheodory Number of a Point . . . 182
Sergey Bereg and Mohammadreza Haghpanah

Characterization and Computation of Feasible Trajectories for an Articulated Probe with a Variable-Length End Segment . . . 189
Ovidiu Daescu and Ka Yaw Teo

Dynamic Products of Ranks . . . 199
David Eppstein

Closest-Pair Queries and Minimum-Weight Queries are Equivalent for Squares . . . 206
Abrar Kazi and Michiel Smid

Parallel topological sweep . . . 214
Ming Ouyang

One Hop Greedy Permutations . . . 221
Don R. Sheehy

Session 3B

A Degree 3 Plane 5.19-Spanner for Points in Convex Position . . . 226
Davood Bakhshesh and Mohammad Farshi

Non-Crossing Matching of Online Points . . . 233
Prosenjit Bose, Paz Carmi, Stephane Durocher, Shahin Kamali and Arezoo Sajadpour

Restricted-Weight Minimum-Dilation Spanners on Three Points . . . 240
Kevin Buchin, Herman Haverkort and Hidde Koerts

Fréchet Distance Between Two Point Sets . . . 249
Maike Buchin and Bernhard Kilgus

Realizing m-uniform four-chromatic hypergraphs with disks . . . 258
Gábor Damásdi and Dömötör Pálvölgyi

Red-Blue Point Separation for Points on a Circle . . . 266
Neeldhara Misra, Harshil M. and Aditi Sethia

Session 4A

A lower bound on the number of colours needed to nicely colour a sphere 273
Péter Ágoston

2048 Without Merging 285
Hugo Akitaya, Erik D. Demaine, Jason S. Ku, Jayson Lynch and Csaba D. Tóth

External Exploration of a Convex Polygon 292
Kyle Clarkson and Will Evans

City Guarding with Limited Field of View 300
Ovidiu Daescu and Hemant Malik

Blind Voronoi Game 312
Omid Gheibi and Hamid Zarrabi-Zadeh

Line Segment Visibility: Theoretical and Experimental Results 317
Jonathan Lenchner and Eli Packer

Session 4B

Some Geometric Applications of Anti-Chains 326
Sariel Har-Peled and Mitchell Jones

Discrete Helly type theorems 332
Frederik Jensen, Aadi Joshi and Saurabh Ray

If You Must Choose Among Your Children, Pick the Right One 336
Brittany Terese Fasy, Benjamin Holmgren, Bradley McCoy and David L. Millman

A Simple Algorithm for kNN Sampling in General Metrics 345
Kirk P. Gardner and Don R. Sheehy

Social Distancing is Good for Points too! 352
Alejandro Flores-Velazco

Tribute to Godfried Toussaint

Erik Demaine*

Over three decades ago, Godfried Toussaint (1944-2019) cofounded the Canadian Conference on Computational Geometry. As a father to the conference and the field in general, his impact and influence was immense. In his honor, I will talk about some of my favorite research that he and I did together, including several past CCCG papers. Possible topics include:

- Godfried's only paper with a counterexample to his own work
- Geometry of musical rhythm, with connections to Euclid's GCD algorithm
- Geometry of sand drawing (ethnomathematics), with some new updates presented here at CCCG 2020
- Machine learning through Voronoi diagrams
- Untangling linkages
- Classic computational geometry, such as guarding polyhedra and polyhedronizing point sets
- Godfried's supercollaborative model of doing research

*Massachusetts Institute of Technology, USA, edemaine@mit.edu

Minimum Ply Covering of Points with Convex Shapes

Ahmad Biniaz* Zhikai Lin†

Abstract

Introduced by Biedl, Biniaz, and Lubiw (CCCG 2019), the *minimum ply covering* of a point set P with a set S of geometric objects in the plane asks for a subset S' of S that covers all points of P while minimizing the maximum number of overlapping objects at any point in the plane (not only at points of P). This problem is NP-hard and cannot be approximated by a factor better than 2. Biedl et al. studied this problem for objects that are unit squares or unit disks. They present 2-approximation algorithms that run in polynomial time when the optimum objective value is bounded by a constant. We generalize this result and obtain a 2-approximation algorithm for any fixed-size convex shape. The new algorithm also runs in polynomial time if the optimum objective value is bounded.

1 Introduction

The problem of covering clients with antennas has been well studied in wireless networks [1, 3, 4, 5, 7, 9, 11]. Covering clients by placing new antennas can cause interference (this happens when more than one antenna cover the same region). Covering clients and—at the same time—reducing interference is a big challenge in wireless networks. In this paper we study a geometric problem that addresses this issue.

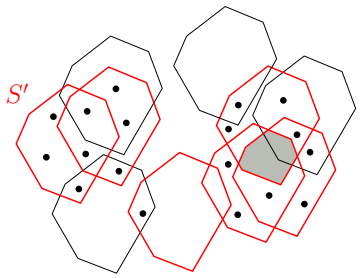


Figure 1: The ply of S' (shown in red) is 3.

Let P be a set of points and S be a set of geometric objects, both in the plane; each element of P represents a client and each object in S represents a coverage region of an antenna. We want to find a subset S' of S that covers all points in P and minimizes the maximum number of overlapping objects at any point in the plane. The *ply* of S' is the maximum number of overlapping objects of S' over all points of the plane. In other words,

$$ply(S') = \max_{p \in \mathbb{R}^2} |\{O \in S' : p \in O\}|.$$

See Figure 1 for an illustration. The term ply was used earlier by Eppstein and Goodrich [6]. With this definition, our goal is to find a subset of S, with minimum ply, that covers P. This problem is introduced by Biedl et al. [2], and it is known as the *minimum ply covering* (MPC). We denote an instance of the MPC problem by (P, S). The MPC problem has the same flavor as the geometric *minimum membership set cover* (MMSC) problem which asks for a subset S' of S that covers all points of P and minimizes the maximum number of overlapping objects only at points of P. Notice that the MPC problem minimizes the maximum number of overlapping objects over all points of the plane.

Erlebach and van Leeuwen [7] showed that the geometric MMSC problem is NP-hard for axis-aligned unit squares and unit disks, and it cannot be approximated by a factor better than 2 in polynomial time. According to Biedl et al. [2] the MPC problem is also NP-hard for axis-aligned unit squares and unit disk, and it cannot be approximated by a ratio better than 2. They presented factor-2 approximation algorithms for the MPC problem with unit squares and unit disks. Their algorithms run in linear time if the optimal ply is bounded by a constant.

In this paper we study the MPC problem for general convex shapes. Let C be an arbitrary convex polygon in the plane. The objects in S are translations of C. We present an algorithm that finds a subset S' of S, with ply at most 2ℓ, that covers all points of P, where ℓ is the optimal ply. In other word, we present a 2-approximation algorithm for the problem instance (P, S). The following theorem summarizes our result.

Theorem 1 *There exists a 2-approximation algorithm for the minimum ply covering of points with fixed-size convex polygons that runs in polynomial-time when the optimal objective value is bounded by a constant.*

Our algorithm is a generalization of the algorithm of Biedl et al. [2]. We first give an overview of their algorithm, and then we show how to extend it to work for any convex shape.

*School of Computer Science, University of Windsor, ahmad.biniaz@gmail.com

†School of Computer Science, University of Windsor, lin12v@uwinsdor.ca

2 Algorithm of Biedl, Biniaz, and Lubiw

We describe their 2-approximation algorithm for unit squares. The main idea of their unit disks algorithm is similar to that of unit squares. Let S be a set of axis-aligned squares of side length 1. Recall that P is a set of points in the plane. To solve the instance (P, S), the algorithm partitions the plane into horizontal slabs of height 2. Let $H_1, H_2, \ldots$ denote these slabs from bottom to top. Let P_j be the points of P in H_j and let S_j be the squares of S that intersect H_j, as in Figure 2. Every square intersects at most two (neighboring) slabs and thus it can appear in at most two sets S_j. The idea is to first solve the MPC problem for each slab H_j optimally, i.e., to solve (P_j, S_j) instances. Let S'_j be an optimal solution for slab H_j. Then take S' as the union of all solutions S'_j. The set S' is a 2-approximate solution for the original problem because every square can appear in at most two S'_j.

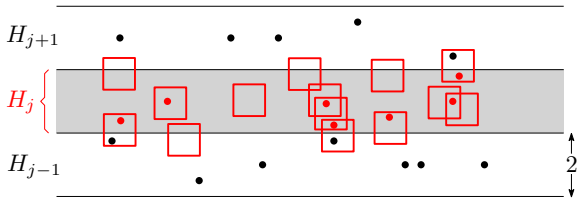


Figure 2: Partitioning the plane into slabs. Red points belong to P_j and red squares belong to S_j.

Assume that the optimal ply is at most ℓ. To solve the (P_j, S_j) instance, partition H_j into vertical strips by vertical lines through the leftmost and rightmost points of all squares.[1] Let $t_1, t_2, \ldots, t_k$ denote these strips from left to right. The following observation plays an important role in the design of the algorithm: *if S^*_j is a solution of (P_j, S_j) with ply at most ℓ, then each strip t_i is intersected by at most 3ℓ squares of S^*_j.*[2] This observation is used to construct a directed acyclic graph G such that any path from the source to the destination in G corresponds to a solution of (P_j, S_j). The graph G is constructed as follows.

For every strip t_i, define a vertex set V_i as follows. Consider every subset $Q \subseteq S_j$ containing at most 3ℓ squares that intersect t_i. Add a vertex $v_i(Q)$ to V_i if (i) the ply of Q is at most ℓ, and (ii) the squares in Q cover all points of P_j that lie in t_i. Notice that no square intersects the strips t_1 and t_k. Thus the set V_1 has exactly one vertex $v_1(\emptyset)$ which is called the "source", and the set V_k has exactly one vertex $v_k(\emptyset)$ which is called the "sink". The vertex set of G is the union of all vertex sets V_i.

[1] In case of squares, the vertical line through the leftmost (resp. rightmost) point is essentially the line through the left (resp. right) side of square.

[2] This number is at most 8ℓ for unit disks [2].

The edges of G are defined base on the following observation. Imagine we scan an optimal solution S^*_j from left to right. While moving from a strip t_i to t_{i+1} either one square stops at their boundary, or one square starts at their boundary, or the squares that intersect t_{i+1} are the same as those intersect t_i. Based on this, we add a directed edge from every vertex $v_i(Q)$ in V_i to every vertex $v_{i+1}(Q')$ in V_{i+1} if one of the following conditions hold

1. $Q' = Q$ as in Figure 3(a), or
2. $Q' = Q \setminus \{q\}$, where q is the square whose right side is on the left boundary of t_{i+1} as in Figure 3(b), or
3. $Q' = Q \cup \{q\}$, where q is the square whose left side is on the left boundary of t_{i+1} as in Figure 3(c).

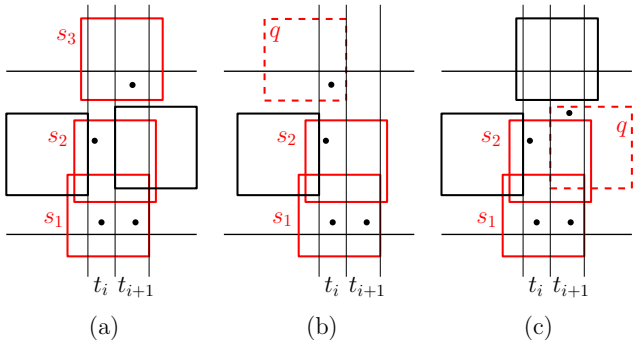


Figure 3: Constructing edges of G where (a) $Q = Q' = \{s_1, s_2, s_3\}$ (b) $Q = \{s_1, s_2, q\}$ and $Q' = \{s_1, s_2\}$ (c) $Q = \{s_1, s_2\}$ and $Q' = \{s_1, s_2, q\}$.

Let δ be any path from the source $v_1(\emptyset)$ to the sink $v_k(\emptyset)$. The union of all sets Q corresponding to the vertices of δ is a solution of (P_j, S_j). The running time of this algorithm for one slab H_j is $O\left((\ell + |P_j|) \cdot |S_j|^{3\ell+1}\right)$, and for all slabs is $O\left((\ell + n) \cdot (2m)^{3\ell+1}\right)$ where $n = |P|$ and $m = |S|$; see section 3.1 for more details. If ℓ is bounded by a constant then the running time is polynomial. The main ingredient to achieve this running time is the fact that the number of squares of any optimal solution S^*_j that intersect any strip t_i is bounded by a constant multiple of ℓ. We are going to obtain a similar fact for all convex shapes, and then extend the algorithm to work for any convex shape.

3 Minimum ply covering with convex shapes

Let P be a set of n points in the plane, and let S be a set of m objects that are translations of the same convex polygon C, as in Figure 1. We show how to find a subset S' of S, with ply at most 2ℓ, that covers all points of P, where ℓ is the optimal ply. In other words, we present a 2-approximation algorithm for the problem instance

(P, S). The algorithm takes polynomial time when ℓ is a constant.

Before proceeding to the algorithm we introduce some terminology. A pair of rectangles (r, R) is called *homothetic* if they are parallel and have the same aspect ratio (r and R need not be axis-parallel). A homothetic pair (r, R) is an *approximating pair* for C if $r \subseteq C \subseteq R$, that is, r is enclosed in C and C is enclosed in R; see Figure 4. Let $\lambda(r, R)$ be the smallest ratio of the length of R to the length of r, over all convex shapes. Pólya and Szegö [12] showed that for every convex shape there exists an approximating pair (r, R) with $\lambda(r, R) \leqslant 3$. Schwarzkopf et al. [13] and Lassak [10] improved this upper bound to 2.[3] For any convex polygon C, an approximating pair of ratio at most 2, can be computed in $O(\log^2 |C|)$ time if the vertices of C are given as a sorted array [13]. The upper bound 2 for $\lambda(r, R)$ is the best possible because for a triangle the length of smallest enclosing rectangle is at least 2 times the length of its largest enclosed homothetic rectangle.

Let (r, R) be an approximating pair for our convex polygon C such that $\lambda(r, R) \leqslant 2$. For simplicity we assume that $\lambda(r, R) = 2$ (this can be achieved by enlarging R or by shrinking r). After a suitable rotation and scaling assume that the longer side of R is vertical and its length is 1. Let α denote the length of the smaller side of R after scaling, as in Figure 4. In this setting the side lengths of r are $1/2$ and $\alpha/2$.

As before, we partition the plane into horizontal slabs of height 2, and then for every slab H_j we solve the problem instance (P_j, S_j) optimally. To solve this instance we partition H_j into vertical strips $t_1, \ldots, t_k$ by vertical lines through the leftmost and the rightmost points of every object in S_j. To construct the corresponding directed acyclic graph G we use the following lemma. This lemma, which is our main technical result, uses the concept of approximating pair of rectangles.

Lemma 2 *Let $S_j^* \subseteq S_j$ be any solution with ply at most ℓ for the problem instance (P_j, S_j). Then any strip t_i is intersected by at most 12ℓ objects in S_j^*.*

Proof. After a suitable translation assume that H_j has y-range $[0, 2]$, and that the y-axis lies in t_i, as in Figure 4. Consider any object C in S_j, and let (r, R) be its approximating pair. We refer to the bottom-left corner of r as the *representative point* of C, and denote it by c. Let h and w be the distances from c to the bottom and left sides of R, respectively. Then the distances from c to the top and right sides of R are $1 - h$ and $\alpha - w$, as in Figure 4. Consider the rectangle F with bottom-left corner $(w - \alpha, h - 1)$ and top-right corner $(w, 2 + h)$. The length of F is 3 and its width is α. Cover F by 12 instances of r, say $r_1, r_2, \ldots, r_{12}$. Denote the top-right corner of each r_k by p_k; these corners are marked by green points in Figure 4.

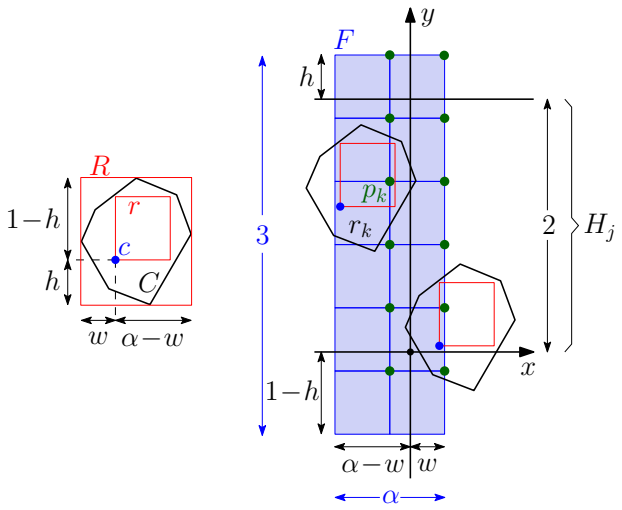


Figure 4: Illustration of the proof of Lemma 2.

Assume that C intersects the strip t_i. Then C intersects the y-axis because vertical strips are defined by vertical lines through leftmost and rightmost points of objects in S_j. In this setting, our definition of h, w, and F imply that the representative c of C must lie in rectangle F. Since F is covered by instances of r, the point c must lie in one of these instances, say r_k. In this case the enclosed rectangle r of C contains p_k, and so does C. Thus, each object in S_j that intersects t_i contains at least one of the points $p_1, \ldots, p_{12}$. Since S_j^* has ply at most ℓ, each point p_k lies in at most ℓ objects of S_j^*. Therefore, at most 12ℓ objects of S_j^* intersect t_i. □

We use Lemma 2 to construct a directed acyclic graph G, analogous to that of [2]. The main difference between the two constructions is in the definition for vertex set V_i of each strip t_i: for every subset Q of at most 12ℓ squares that intersect t_i we introduce a vertex $v_i(Q)$ if (i) the ply of Q is at most ℓ, and (ii) its squares cover all points in t_i. The edges of G are defined as before. Any path from the source to the sink in G corresponds to a solution of (P_j, S_j)—this claim, which is proved in [2] for squares and circles, holds for any convex shape and in particular for C. This is the end of the algorithm and its correctness proof.

3.1 Time complexity

The running time analysis is analogous to that of [2] for squares, and thus we keep it short. Set $n_j = |P_j|$ and $m_j = |S_j|$. Then the number of strips is $k = 2m_j + 1$. The number of vertices in every set V_i is $O\left(m_j^{12\ell}\right)$. Therefore the total number of vertices of G is at most $k \cdot O\left(m_j^{12\ell}\right) = O\left(m_j^{12\ell+1}\right)$. Since every vertex has at most three outgoing edges, the number of edges of G is also

[3] A similar ratio is also obtained for pairs of ellipses that approximate convex shapes [8].

$O\left(m_j^{12\ell+1}\right)$. By an initial sorting of the points of P_j and the objects of S_j with respect to the y-axis, conditions (i) and (ii) can be verified in $O\left(|C| \cdot (\ell + n_j)\right)$ time for each subset Q, where $|C|$ is the number of vertices of C. Therefore, it takes $O(|C| \cdot (\ell + n_j) \cdot m_j^{12\ell+1})$ time to construct G. A path from the source to the sink in G can be found in time linear in the size of G. Thus, the total running time to solve the problem instance (P_j, S_j) is $O(|C| \cdot (\ell + n_j) \cdot m_j^{12\ell+1})$. Since every point of P belongs to one slab and every object of S belongs to at most two slabs, the running time of the entire algorithm—for all slabs—is $O(|C| \cdot (\ell + n) \cdot (2m)^{12\ell+1})$, which is polynomial when ℓ is bounded by a constant.

4 Conclusion

We generalized the 2-approximation algorithm of Biedl et al. [2] for the MPC problem to work for any convex shape. A natural question is to verify if there are polynomial-time O(1)-approximation algorithms for the MPC problem when the objective value is not necessarily a constant.

References

[1] M. Basappa, R. Acharyya, and G. K. Das. Unit disk cover problem in 2D. *Journal of Discrete Algorithms*, 33:193–201, 2015.

[2] T. Biedl, A. Biniaz, and A. Lubiw. Minimum ply covering of points with disks and squares. In *Proceedings of the 31st Canadian Conference on Computational Geometry (CCCG)*, pages 226–235, 2019.

[3] A. Biniaz, P. Liu, A. Maheshwari, and M. H. M. Smid. Approximation algorithms for the unit disk cover problem in 2D and 3D. *Comput. Geom.*, 60:8–18, 2017. Also in CCCG'15.

[4] P. Carmi, M. J. Katz, and N. Lev-Tov. Covering points by unit disks of fixed location. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC)*, pages 644–655, 2007.

[5] G. K. Das, R. Fraser, A. López-Ortiz, and B. G. Nickerson. On the discrete unit disk cover problem. *Int. J. Comput. Geometry Appl.*, 22(5):407–420, 2012. Also in WALCOM'11.

[6] D. Eppstein and M. T. Goodrich. Studying (nonplanar) road networks through an algorithmic lens. In *Proceedings of the 16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS*, 2008.

[7] T. Erlebach and E. J. van Leeuwen. Approximating geometric coverage problems. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1267–1276, 2008.

[8] F. John. Extremum problems with inequalities as subsidiary conditions. *Studies and Essays Presented to R. Courant on his 60th Birthday, January 8, 1948, Interscience, New York*, pages 187–204, 1948.

[9] F. Kuhn, P. von Rickenbach, R. Wattenhofer, E. Welzl, and A. Zollinger. Interference in cellular networks: The minimum membership set cover problem. In *Proceedings of the 11th International Computing and Combinatorics Conference (COCOON)*, pages 188–198, 2005.

[10] M. Lassak. Approximation of convex bodies by rectangles. *Geometriae Dedicata*, 47:111–117, 1993.

[11] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[12] G. Pólya and G. Szegő. *Isoperimetric Inequalities in Mathematical Physics*. Annals of Mathematics Studies 27, Princeton University Press, 1951.

[13] O. Schwarzkopf, U. Fuchs, G. Rote, and E. Welzl. Approximation of convex figures by pairs of rectangles. *Comput. Geom.*, 10(2):77–87, 1998. Also in STACS'90.

Convex Hull Complexity of Uncertain Points

Hongyao Huang* Benjamin Raichel*

Abstract

An uncertain point set U is a collection of compact regions in the plane, and a realization of U is any point set determined by selecting one point from each set in U. Here we consider the problem of determining the realization whose convex hull has the minimum number of vertices possible. We prove that when U is a set of n parallel line segments then the problem can be solved in $O(n^3)$ time, but when the line segments can have arbitrary orientations then the problem is NP-Complete.

1 Introduction

Uncertainty in computational problems has received significant attention in recent years, as many real world inputs are inherently noisy. Such problems have been particularly well studied within computational geometry, as uncertainty naturally arises when for example collecting locational data from the physical world.

In this paper we consider the complexity of the convex hull, one of the most fundamental geometric structures, in the context of uncertainty. Specifically, given an uncertain point set $U = \{u_1, \ldots, u_n\}$, where each u_i is a compact region in the plane, a realization of U is any point set $P = \{p_1, \ldots, p_n\}$ such that $p_i \in u_i$ for all i. Here we consider finding the realization of U whose convex hull has the minimum number of vertices.

To the best of our knowledge, our paper is the first to consider the minimum complexity of the convex hull in such uncertain settings when measured by the number of vertices. Previous papers have considered the problem when complexity is measured by perimeter or area. Rappaport [20] computed the minimum perimeter convex hull for line segments with a constant number of orientations in near linear time. Mukhopadhyay *et al.* [19] computed the minimum area convex hull for parallel lines in near linear time. Subsequently, Löffler and van Kreveld [18] did an extensive study on finding the realization which either minimized or maximized the area or perimeter of the convex hull, where different algorithmic or hardness results were given depending on the shape of the uncertain regions.

Various other geometric structures have also been considered in uncertain settings, such as bounding boxes [17], Delaunay triangulations [3, 16], Voronoi diagrams [5, 7, 12, 15], terrains [6, 9], and more. When the uncertain points have associated probabilities, the expected complexity of the convex hull was previously studied (see [11] and references therein). Related questions concerning the convex hull have also been considered, such as computing the probability a given query point is contained in the convex hull [1], or computing the most likely convex hull of probabilistic points [21]. More generally, Jørgensen *et al.* [13] considered the distributions of various geometric quantities in probabilistic settings.

Our problem also relates to the traversal problem, where given a set of convex regions in the plane, one seeks a polygonal chain with some property which stabs all the regions. When the regions are disjoint and ordered, Guibas *et al.* [10] gave efficient algorithms to compute the minimum link polygonal chain stabbing the objects in order. When the objects are parallel line segments, Goodrich and Snoeyink [8] gave a near linear time algorithm to compute a convex stabber if it exists, that is a selection of a single point from each segment such that the resulting set is in convex position. For line segments with general orientations, Arkin *et al.* [2] proved that determining the existence of such a convex stabber is NP-hard. More recently, for the case when the line segments have a constant number of orientations, Keikha *et al.* [14] gave a polynomial time algorithm to determine if there is a convex stabber which stabs at least k segments.

Our Contribution. Our main result is an $O(n^3)$ time algorithm to compute the realization whose convex hull has the fewest vertices when the uncertain regions are parallel line segments. Without loss of generality, for this case we can assume the segments are all vertical. The behavior of our minimization problem differs from the previously studied minimization problems for perimeter or area [19, 20], and instead behaves more similarly to the problem of maximizing the area, for which Löffler and van Kreveld [18] gave an $O(n^3)$ time algorithm. There the authors argued one can assume each segment is realized either at its top or bottom endpoint. This is no longer true for our problem, however, we can argue that other than the leftmost and rightmost segment, one can assume all segments defining vertices of the convex hull are realized either at their top or bottom endpoint. The differences between these two state-

*Department of Computer Science, University of Texas at Dallas, {hhuang, benjamin.raichel}@utdallas.edu. Partially supported by a NSF CAREER Award 1750780.

ments, makes achieving the same $O(n^3)$ running time for our problem more challenging, particularly when it comes to determining the leftmost and rightmost points. Related challenges arise in the problem of maximizing the number of stabbed segments in a convex traversal, considered by Keikha *et al.* [14], for which the authors give an $O(n^6)$ time algorithm. The $O(n^3)$ running time of our algorithm thus compares favorably, though such a comparison is limited as the problems differ.

We complement our algorithmic result for parallel line segments, by proving that the problem is NP-Complete when the line segments can have arbitrary orientations. Our reduction is inspired by the NP-hardness proof in [18]. However, as our problem is a minimization problem and theirs a maximization problem, additional points must be added to keep the gadgets from collapsing inwards to a trivial solution.

1.1 Preliminaries

We follow the uncertain point model of previous papers such as [18], where an uncertain point is modeled by an uncertain region u, which is any compact subset of the plane. For a set of uncertain regions $U = \{u_1, u_2, ..., u_n\}$, a *realization* of U is any point set $P = \{p_1, \ldots, p_n\}$ such that $p_i \in u_i$ for all i. Let $Real(U)$ denote the set of all possible realizations of U.

Given a point set P, let $CH(P)$ denote the convex hull of P, and let $|CH(P)|$ denote the number of vertices of the convex hull, where a vertex of $CH(P)$ is any point $q \in P$ such that $q \notin CH(P \setminus \{q\})$.

Problem 1 *Given a set $U = \{u_1, u_2, ..., u_n\}$ of uncertain regions, compute* $\arg\min_{P \in Real(U)} |CH(P)|$.

Throughout we will use the following basic polygonal chain definitions.

Definition 2 *A* polygonal chain *is an ordered sequence of points in the plane $P = \{p_1, \ldots, p_n\}$. P is* monotone *(resp. reverse monotone) if for all $1 \leq i < n$, p_{i+1} has larger (resp. smaller) x-coordinate than p_i. P is* convex *if for all $1 < i < n$, p_{i-1}, p_i, p_{i+1} defines a right turn, that is p_{i+1} lies to the right of the line segment $\overline{p_{i-1}p_i}$ when directed from p_{i-1} to p_i. If P is convex and monotone (resp. reverse monotone) then it is called a* top chain *(resp.* bottom chain*). P is* simple *if for all $i < j$, $\overline{p_ip_{i+1}}$ and $\overline{p_jp_{j+1}}$ do not intersect, except at p_{i+1} when $j = i + 1$. (Bottom and top chains are simple.)*

Let Q be a point set, and let p_l and p_r respectively be the leftmost and rightmost points in Q. $CH(Q)$ is described by a simple closed convex polygonal chain of its vertices, which is composed of a top chain from p_l to p_r followed by a bottom chain from p_r to p_l.

For a point p in the plane, let $p.x$ and $p.y$ denote its x and y coordinate, respectively. Let $P = \{p_1, \ldots, p_n\}$ be a monotone polygonal chain, and let q be any point in the plane such that $p_1.x \leq q.x \leq p_n.x$. Let i be the index such that the vertical line through q intersects the segment $\overline{p_ip_{i+1}}$. Then we say q lies *below* (resp. *above*) the monotone chain P if it lies below (resp. above) or on the segment $\overline{p_ip_{i+1}}$.

2 Vertical Line Segments

In this section, we give a polynomial time algorithm for Problem 1, when U is a set of vertical line segments S. More generally, the algorithm works for any set of parallel line segments, as rotation does not change $|CH(P)|$. Throughout we let $S = \{s_1, \ldots, s_n\}$ denote a set of vertical line segments, where for simplicity we assume no two segments lie on the same vertical line, and the segments are ordered such that for $i < j$ segment s_i lies to the left of s_j. For a segment s_i, we use s_i^+ to denote its top endpoint, and s_i^- to denote its bottom endpoint.

Definition 3 *Call a monotone polygonal chain $P = \{p_1, \ldots, p_m\}$ a* positive chain *with respect to S if, $p_1 \in s_1$, $p_m \in s_n$, and for all $1 < i < m$, $p_i = s_j^+$ for some j. Similarly, define negative chains.*

Lemma 4 *When U is a set of vertical line segments S, there is an optimal solution to Problem 1, such that the top chain of the convex hull is a positive chain and the bottom chain is a negative chain.*

Proof. Consider any set $R \in Real(S)$, and let $T = \{t_1, \ldots, t_m\}$ be the top chain of $CH(R)$. Let $t_i \in T$ be any vertex of the top chain other than t_1 and t_m, and let t_i^+ be the upper endpoint of the segment which generated t_i. Let $H_{old} = CH(R)$ and $H_{new} = CH((R \setminus t_i) \cup t_i^+)$. We now argue that $|H_{new}| \leq |H_{old}|$. This will prove the lemma, as one can then iteratively move each vertex remaining on the top chain to its upper segment endpoint until all top chain vertices are at their upper segment endpoints, without ever increasing the number of top chain vertices. A symmetric argument applies to the bottom chain.

Observe that H_{old} and H_{new} are convex hulls of the same set of points except where t_i has been exchanged for t_i^+. This implies that if we can argue that $H_{old} \subseteq H_{new}$ then any vertex of H_{new} (other than t_i^+) must be a vertex of H_{old} and thus $|H_{new}| \leq |H_{old}|$ as desired. Note that $CH(R \setminus t_i) \subseteq H_{new}$, thus to argue $H_{old} \subseteq H_{new}$, it suffices to argue $t_i \in H_{new}$. To this end, note that t_{i-1} and t_{i+1} exist as $1 < i < m$. So let z be the point on $\overline{t_{i-1}t_{i+1}}$ lying directly below t_i, i.e. with the same x coordinate, and note that z is well defined as t_{i-1}, t_i, and t_{i+1} are consecutive on the upper chain T (which is convex monotone). Moreover, $z \in H_{new}$ as H_{new} contains both t_{i-1} and t_{i+1}. As t_i^+ lies directly above t_i and z, we have that $t_i \in \overline{z\, t_i^+} \subseteq H_{new}$, proving the lemma. □

The above lemma suggests a natural dynamic programming strategy. Process the segments in S from left to right, where at each segment if we decide it corresponds to a hull vertex then we take its top or bottom endpoint. If it does not correspond to a hull vertex, then by mainitaining appropriate information about the previously selected hull vertices, we will enforce that the segment intersects the final hull, implying its realization can be inside the hull.

Intuitively the structure we wish to maintain is the top and bottom chains of the optimal convex hull. First, observe that these chains cannot be computed independently as selecting a vertex for the top chain affects whether it can or needs to be selected for the bottom chain. Thus as we go from left to right we will remember the last vertex selected from both the top and the bottom chains. Enforcing convexity, however, would require remembering the previous edge not the previous vertex, which would be more expensive. Thus instead we look for positive and negative chains with the fewest vertices, which may not be convex but must satisfy certain properties implied by convexity, and then we use the lemma below to argue these properties are sufficient. (Ultimately one can argue minimal such chains are in fact top and bottom chains, though it is not necessary.)

Definition 5 *Call a pair P^+, P^- of positive and negative chains, a* valid chain pair *if the first and last vertices of P^+ are the same as those of P^-, and for all $1 < i < n$ (i) if $s_i^+ \in P^+$ then $s_i^- \notin P^-$ and if $s_i^- \in P^-$ then $s_i^+ \notin P^+$, (ii) s_i^- lies below P^+, and (iii) s_i^+ lies above P^-.*

The proof of the following is in Appendix A.2.

Lemma 6 *Let P^+, P^- be a valid chain pair. Then $\mathcal{CH}(P^+ \cup P^-)$ intersects all segments in S.*

By Lemma 4 we know that there is an optimal solution to Problem 1 such that the top chain of the convex hull is a positive chain $\hat{P}^+$ and the bottom chain is a negative chain $\hat{P}^-$. Note that all points in this optimal realization of S lie below the top chain and above the bottom chain of the convex hull. This implies that for all $1 < i < n$, s_i^- lies below $\hat{P}^+$ and s_i^+ lies above $\hat{P}^-$, and therefore $\hat{P}^+, \hat{P}^-$ is a valid chain pair. So let P^+, P^- be a valid chain pair minimizing $|P^+|+|P^-|$. By Lemma 6 $\mathcal{CH}(P^+ \cup P^-)$ intersects all segments in S, and thus there is a realization of S whose convex hull vertices all lie in $P^+ \cup P^-$. As clearly $|P^+| + |P^-| \leq |\hat{P}^+| + |\hat{P}^-|$, we have the following.

Corollary 7 *Let P^+, P^- be a valid chain pair which minimizes $|P^+| + |P^-|$ over all valid chain pairs. Then $P^+ \cup P^-$ are the vertices of an optimal solution to Problem 1 on S.*

By the above it thus suffices to compute the minimum sized valid chain pair, which can easily be accomplished using a standard dynamic programming approach. Specifically, the recursive Algorithm 1 maintains the previous vertex selected on the positive chain, s_i^+, and the previous vertex selected on the negative chain, s_j^-, and then tries all possible choices for the next vertex to the right (of both s_i and s_j), which if on segment s_k could be either s_k^+ or s_k^-. Specifically, in order for s_k^+ to be considered as a possible next vertex on the positive chain, by Definition 5, we must require that for all $i < x < k$ that s_x^- lies below the segment $\overline{s_i^+ s_k^+}$. Assume we have a function $\textsc{Positive}(i)$ that computes all such indices. For k to be a valid next index we also require $k > \max\{i, j\}$. Thus if P denotes the set of all possible next positive chain vertex indices, then $P = \textsc{Positive}(i) \cap \{\max\{i, j\} < k < n\}$. Similarly define the set of possible next negative chain vertices $N = \textsc{Negative}(j) \cap \{\max\{i, j\} < k < n\}$, where $\textsc{Negative}(j)$ is defined analogously to $\textsc{Positive}(i)$. Finally, define $\textsc{EndRight}(i, j)$ as the function which returns true if we can extend the current chains directly to the rightmost segment s_n, namely does there exist a point $r \in s_n$ such that s_x^- lies below the segment $\overline{s_i^+ r}$ for $i < x < n$ and s_x^+ lies above the segment $\overline{s_j^- r}$ for $j < x < n$.

Algorithm 1 Recursive Algorithm for Problem 1

Output: Min number of remaining valid chain pair vertices or ∞ if no solution, given the previous positive and negative chain vertices were s_i^+ and s_j^-.

```
1:  function MinCH(i, j)
2:      P ← Positive(i) ∩ {max{i, j} < k < n}
3:      N ← Negative(j) ∩ {max{i, j} < k < n}
4:      value ← ∞
5:      for k ∈ P do
6:          value ← min{value, 1 + MinCH(k, j)}
7:      for k ∈ N do
8:          value ← min{value, 1 + MinCH(i, k)}
9:      if EndRight(i, j) then
10:         value = 1
11:     return value
```

First we argue that when the leftmost segment s_1 is a single point (or equivalently we know the point to select on segment s_1), then Algorithm 1 can be used to to solve Problem 1 in cubic time. Afterwards, we argue how to remove this assumption on s_1 while maintaining the same running time.

Theorem 8 *For a set $S = \{s_1, \ldots, s_n\}$ of vertical segments, where the leftmost segment s_1 is a single point, Problem 1 can be solved in $O(n^3)$ time.*

Proof. Assuming that $\textsc{Positive}(i)$, $\textsc{Negative}(j)$, and $\textsc{EndRight}(i, j)$ all work as described

above then Algorithm 1 sets $\text{MinCH}(i,j) = 1 + \min\{\min_{k\in P} \text{MinCH}(k,j), \min_{k\in N} \text{MinCH}(i,k)\}$ or 1 if we can connect directly to the rightmost segment, where P and N are respectively the sets of all possible next positive and negative chain vertices. Thus $\text{MinCH}(1,1)+1$ computes the size of a minimum cardinality valid chain pair, which by Corollary 7 corresponds to an optimal solution to Problem 1. Note because s_1 is a single point, $s_1 = s_1^+ = s_1^-$, thus all subroutine calls are well defined, and $\text{MinCH}(1,1)$ will start the valid chain pairs on the same point as required, where this point is counted by the $+1$.

As i and j both range over $O(n)$ possible values, this recursive algorithm can be turned into a dynamic program with a table of total size $O(n^2)$. Assuming $\text{Positive}(i)$, $\text{Negative}(j)$, and $\text{EndRight}(i,j)$ all run in $O(n)$ time, then each table entry takes $O(n)$ time to compute as outside those subroutines the code consists of constant time operations and two disjoint for loops going over P and N. This then gives an $O(n^3)$ running time overall as claimed. Thus what remains is to describe how to implement $\text{Positive}(i)$, $\text{Negative}(j)$, and $\text{EndRight}(i,j)$ in linear time.

First, we describe how to compute $P' = \text{Positive}(i)$ in linear time, from which one can then easily compute $P = P' \cap \{\max\{i,j\} \le k < n\}$. Fix an index $k > i$, and let $X = \{x \mid i < x < k\}$. Consider the ray from s_i^+ pointing vertically downwards. Each point s_x^- for $x \in X$ determines an angle with this ray, when rotating the ray counterclockwise. Let s_{max}^- be the point with the largest such angle from the index set X. If s_k^+ lies above the line supporting the segment $\overline{s_i^+ s_{max}^-}$, then s_{max}^- and hence all s_x^- for $x \in X$ lie below $\overline{s_i^+ s_k^+}$ as required for k to be in P'. Conversely, if s_k^+ lies below the line supporting the segment $\overline{s_i^+ s_{max}^-}$, then s_{max}^- would not lie below the line $\overline{s_i^+ s_k^+}$ and so $k \notin P'$. Thus if our algorithm maintains s_{max}^- as we increment k then in constant time we can check if $k \in P'$, an moreover s_{max}^- can be updated in constant time per iteration by comparing the new bottom endpoint with the previous s_{max}^-. Thus $P' = \text{Positive}(i)$ can be computed in linear time, as shown in Algorithm 2, in Appendix A.1. A similar argument allows us to compute $N' = \text{Negative}(j)$ in linear time as is also shown in Algorithm 2.

Now we describe how to compute $\text{EndRight}(i,j)$ in linear time. Specifically, we seek to determine if there exists a point $r = (r.x, r.y)$ on s_n such that for all $i < x < n$, s_x^- lies below $\overline{s_i^+ r}$, and for all $j < x < n$, s_x^+ lies above $\overline{s_j^- r}$. Note that since s_n is a vertical segment, $r.x$ is fixed, and thus all of these constraints can be written as linear constraints in the one variable $r.y$. In particular, restricting r to lie on s_n means that $s_n^-.y \le r.y \le s_n^+.y$. All other constraints can be written as satisfying a right or a left turn check, each of which is expressible by checking the sign of the determinant of a matrix whose three rows are of the form $(1, s_i^+)$, $(1, r)$, and $(1, s_x^-)$. (Note the cross terms in the determinant are linear in the only variable $r.y$.) Thus we are doing a feasibility check of a linear program with $O(n)$ constraints and one variable. This is easily solved in $O(n)$ time by checking whether the tightest lower bound constraint on $r.y$ lies to the left on the real line of the tightest upper bound constraint on $r.y$. □

Now we remove the assumption that s_1 is a single point. In Appendix A.3 we remark how the optimal starting point must lie in a set of $O(n^2)$ canonical points on s_1, thus leading to an easy $O(n^5)$ time solution by trying our above $O(n^3)$ algorithm on all such points.

Instead of reducing to the single point case, we describe an alternative approach which still runs in $O(n^3)$ time. First, for now assume that the top and bottom chains both have at least one interior vertex (i.e. a vertex not on s_1 or s_n). While we cannot compute $\text{MinCH}(i,j)$ if either $i = 1$ or $j = 1$, we can compute $\text{MinCH}(i,j)$ for all $1 < i,j < n$ in $O(n^3)$ time by the approach of Theorem 8. Let $\text{StartLeft}(i,j)$ be defined similarly to $\text{EndRight}(i,j)$ above, except that it checks in linear time if there is a point l on s_1 such that s_x^- lies below the segment $\overline{ls_i^+}$ for $1 < x < i$ and s_x^+ lies above the segment $\overline{ls_j^-}$ for $1 < x < j$. Let $T = \{1 < i,j < n \mid \text{StartLeft}(i,j) = True\}$, then $3 + \min_{(i,j)\in T} \text{MinCH}(i,j)$ would find the minimum solution over all $1 < i,j < n$ pairs that can connect directly to the leftmost edge s_1 (where the $+3$ counts s_i^+, s_j^-, and the vertex on s_1). Unfortunately, this does not count all possible cases as initially there may be several hull vertices on the top chain interior before the first bottom chain interior vertex, and $\text{MinCH}(i,j)$ assumes s_i^+ and s_j^- are consecutive in the left to right order of vertices on the hull (i.e. we miss cases of the form $\text{MinCH}(1,j)$ and $\text{MinCH}(i,1)$). However, there is a simple way to overcome this issue. Rather than trying to directly connect to the left edge, just compute the minimal chains to the left and then append them to the minimal chains we computed on the right. Specifically, let $\text{MinCHLeft}(i,j)$ be the same as $\text{MinCH}(i,j)$ except that it computes the minimal valid chain pairs to the left (instead of the right) when the previous vertex on the positive chain was s_i^+ and the previous on the negative chain was s_j^-. Note $\text{MinCHLeft}(i,j)$ uses $\text{StartLeft}(i,j)$ instead of $\text{EndRight}(i,j)$, and similarly modifies $\text{Positive}(i)$ and $\text{Negative}(j)$. Therefore we return

$$2 + \min_{1<i,j<n} \{\text{MinCHLeft}(i,j) + \text{MinCH}(i,j)\},$$

where the $+2$ counts the vertices s_i^+ and s_j^-. It is important to note here that $\text{MinCHLeft}(i,j)$ only selects vertices to the left of $\min\{i,j\}$ and $\text{MinCH}(i,j)$ to the

right of $\max\{i,j\}$. That is, both assume there are no hull vertices on segments with indices between $\min\{i,j\}$ and $\max\{i,j\}$, and thus the above approach only works if there exists an index pair i,j from the optimal solution such that s_i^+ and s_j^- are consecutive in the left to right order of vertices on the hull, i.e. there are no hull vertices on segments with indices between $\min\{i,j\}$ and $\max\{i,j\}$. However, it is easy to see this holds by our assumption that there is at least one interior vertex on both the top and bottom chains. As for the running time, observe we can independently precompute all MinCHLeft(i,j) values in $O(n^3)$ time and all MinCH(i,j) values in $O(n^3)$ time, and this dominates the time to compute the above minimum over all index pairs.

So what remains is to handle the case when the optimal solution may not have an interior vertex on either the top or bottom chain. We have the following lemma for the case when the top chain has no interior vertex, the bottom chain case is handled symmetrically. Due to space the proof is in Appendix A.2.

Lemma 9 *For a set S of n vertical segments, the optimal solution to Problem 1 where the top chain of the convex hull is not allowed to have interior vertices can be solved in $O(n^3)$ time.*

By running all cases for whether the bottom or top chain interiors are empty and taking the minimum we thus have the following.

Theorem 10 *For a set S of n vertical segments, Problem 1 can be solved in $O(n^3)$ time.*

In Appendix A.3 we briefly remark how our approach can be extended to ordered axis-aligned rectangles.

3 NP-Hardness for General Segments

We now argue that when the segments in S are not required to be vertical, then Problem 1 is NP-hard. The proof is by reduction from the standard NP-hard problem CNF-SAT. Our reduction closely follows the approach of the NP-hardness proof in [18] for maximizing the area of the convex hull for uncertain line segments. However, our construction requires additional points in the clause and variable gadgets, in large part since our problem involves minimization and theirs maximization.

Let the given instance of CNF-SAT have n variables and m clauses. Call an uncertain segment a certain point if its two endpoints are the same. Consider a circle in the plane, and evenly place a set of certain points, $B = \{b_1, \ldots, b_{n+m}\}$, along this circle. Call these our base points. Observe that if we conceptually remove $\mathcal{CH}(B)$ then we are left with a set of disjoint circular caps, $c_1, \ldots, c_{n+m}$, each bounded some segment $\overline{b_i b_{i+1}}$ and corresponding circular arc from b_i to b_{i+1}, see Figure B.1 in Appendix B. We have one cap for each variable and one for each clause. All remaining uncertain segments we construct will have both their endpoints in the same cap, or in two different caps when those caps correspond to a variable and a clause that contains it. Note that since all segment endpoints will lie in the circle, all base points are always vertices of the convex hull in any realization. This conceptually separates the caps, in the sense if you added a point in one of the caps, the area it adds to the convex hull is confined to that cap. The way we then connect a variable and clause cap is by adding an uncertain segment between them.

First, consider the cap for a given clause L. We create one uncertain segment for each literal in L. All these segments share a common endpoint at the center of the clause cap, the other endpoints are in the caps of the respective variables. Let this common endpoint be denoted e and let b and b' be the base points of the clause cap for L. We add a convex chain of z certain points from b to b' such that all these points are contained in $\mathcal{CH}(\{e, b, b'\})$. Here z an integer value, to be determined shortly, but intuitively we require z be set large enough so that one of the segments adjacent to e, must select e as its realization to cover these z points. See Figure 3.1.

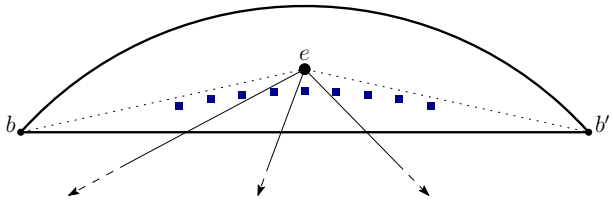


Figure 3.1: Clause cap with common endpoint e. Convex chain of z certain points shown as squares.

Now consider the cap c for some variable x, with corresponding base points b and b'. Within c we add a segment $\overline{tf}$ above and parallel to the segment $\overline{bb'}$, where ultimately selecting t or f will correspond to setting the variable to True or False, respectively. Let l be the maximum over all variables of the maximum of the number of times that variable appears as a positive literal or appears as a negative literal. For the variable x we create a convex chain of l "positive" vertices, P, and a convex chain of l "negative" vertices, N. Specifically, we require (i) every point of P is a vertex of $\mathcal{CH}(P \cup \{b, b', f\})$ (ii) every point of N is a vertex of $\mathcal{CH}(N \cup \{b, b', t\})$ (iii) $N \subset \mathcal{CH}(\{b, b', f\})$, (iv) $P \subset \mathcal{CH}(\{b, b', t\})$, and (v) for any point $v \in \overline{tf}$ if $\mathcal{CH}(\{b, b', v\})$ contains a point of P (resp. N) it does not contain a point of N (resp. P). See Figure 3.2. Recall that for each literal occurrence of x we created an uncertain segment with one end fixed at the corresponding clause. We now make the other end of the uncertain segment a unique point in P or N, depending on whether it appeared as a positive or

negative literal in the clause. We place a certain point at any unused points in P or N. Finally, for each point u in either P or N, we create a small convex chain of z certain points R_u just below it, such that all the points in R_u are contained in $\mathcal{CH}(\{b, b', u\})$ and none are contained $\mathcal{CH}(\{b, b', u'\})$ for any other point u' in either P or N.

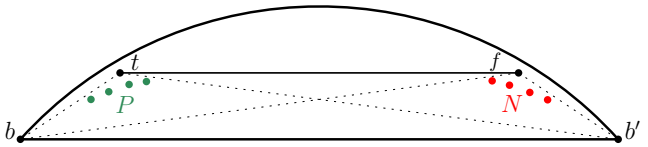


Figure 3.2: Variable cap. Convex chains of z certain points below points in P and N, as well as uncertain segments adjacent to P and N, are not shown.

To argue correctness of the reduction, first suppose there is satisfying assignment to the given CNF-SAT instance. In this case we argue there is a realization of our uncertain segments with $\leq 2m+(l+2)n$ vertices on the convex hull. Specifically, for each variable x, if $x = True$ then in the cap for x we select t for the segment $\overline{tf}$, for each segment adjacent to a point $u \in N$ we select u, and for each segment adjacent to a point in P we select the opposite (i.e. clause) endpoint of the segment. Note that by construction $\mathcal{CH}(N \cup \{b, b', t\})$ contains both P and all the convex chains R_u that we added for each u in P or N, and thus in this case only t and the l points in N are vertices of the convex hull within this cap. Similarly, if $x = False$, we select f for the segment $\overline{tf}$, for each segment adjacent to a point $u \in P$ we select u, and for each segment adjacent to a point in N we select the opposite (i.e. clause) endpoint of the segment. Again, in this case only f and the l points in P are vertices of the convex hull within this cap. On the other hand, for the clause caps, observe that because this was a satisfying assignment for the CNF-SAT instance, we must have selected the common end point in each clause cap for at least one of its adjacent segments. Since for each clause, with common endpoint e and base points b and b', the convex chain of certain points in its cap is contained in $\mathcal{CH}(\{e, b, b'\})$, the number of vertices on the convex hull from this cap is just 1. Thus the total contribution from all clause and variable caps is $m + (l + 1)n$, and since the $n + m$ base points are always on the hull, we thus have $\leq 2m + (l + 2)n$ vertices as claimed.

Now suppose there is no satisfying assignment for the given CNF-SAT instance. In this case we argue that by setting the parameter z to be large enough,the convex hull of any realization has $> 2m + (l + 2)n$ vertices. Specifically, if $z = 2m + (l + 2)n + 1$, then clearly if any one of the chains with z certain points is entirely on the hull then the realization has $> 2m + (l + 2)n$ vertices. Define E as the set of all uncertain segments adjacent to points in P or N from any variable cap, but realized outside the corresponding variable cap. Consider one of the chains with z certain points in some clause cap with common end point e. In order for this chain to not entirely appear on the hull, at least one of the uncertain segments adjacent to e must be in E, and in particular must have its realization somewhere on the e side of the chain. In a minimal solution it can be assumed to be at the point e itself, since as discussed above placing it at e means this clause cap only contributes one vertex, and clearly this cap must contribute at least one vertex in any realization. Now consider a variable cap with base points b and b'. By condition (v) from above, for any point $v \in \overline{tf}$ if $\mathcal{CH}(\{b, b', v\})$ contains a point of P (resp. N) it does not contain a point of N (resp. P). Thus in a minimal solution we can assume $\overline{tf}$ is realized at either t or f. Suppose it is realized at t (the f case is symmetric), and recall that N is not in $\mathcal{CH}(\{t, b, b'\})$. Thus by the same argument as for the clause caps, for every uncertain segment adjacent to a point $u \in N$, a minimal solution can be assumed to select u as the realization, as the chain of z points R_u must be covered. (Recall if u has no adjacent segment we already placed a certain point there.) More generally, in order for a solution to have $< z$ vertices on the convex hull, for any variable v, all uncertain segments that it contributes to E are either all adjacent to points in P (when t is selected) or all adjacent to points in N (when f is selected). So consider the collection of all t and f endpoints chosen for all variable caps in a minimal solution, which thus determines which uncertain segments can fall in E. This collection can be viewed as a variable assignment for the given CNF-SAT instance, and as this instance is not satisfyable, some clause in this assignment evaluates to false. However, this implies that for some common endpoint e in some clause cap, there are no segments adjacent to e that are in E, and hence the number of vertices on the hull is at least $z = 2m + (l + 2)n + 1$.

Thus if we can decide whether there is a realization with $\leq 2m + (l + 2)n$ convex hull vertices, then we can decide the corresponding CNF-SAT instance. Also, it is easy to see that the above uncertain segments can be constructed such that all endpoints are rational points of polynomial complexity (see [18]). Thus we have the following theorem for the decision version of Problem 1.

Theorem 11 *Given a set S of n uncertain segments and an integer k, the problem of determining whether there is a realization of S with $\leq k$ vertices on the convex hull is NP-Complete.*

References

[1] P. K. Agarwal, S. Har-Peled, S. Suri, H. Yildiz, and W. Zhang. Convex hulls under uncertainty. *Algorithmica*, 79(2):340–367, 2017.

[2] E. M. Arkin, C. Dieckmann, C. Knauer, J. S. B. Mitchell, V. Polishchuk, L. Schlipf, and S. Yang. Convex transversals. *Comput. Geom.*, 47(2):224–239, 2014.

[3] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

[4] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.

[5] A. Driemel, S. Har-Peled, and B. Raichel. On the expected complexity of voronoi diagrams on terrains. *ACM Trans. Algorithms*, 12(3):37:1–37:20, 2016.

[6] A. Driemel, H. Haverkort, M. Löffler, and R. Silveira. Flow computations on imprecise terrains. *Journal of Computation Geometry (JoCG)*, 4(1):38–78, 2013.

[7] R. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. pages 326–333, 1989.

[8] M. T. Goodrich and J. Snoeyink. Stabbing parallel segments with a convex polygon. *Computer Vision, Graphics, and Image Processing*, 49(2):152–170, 1990.

[9] C. Gray, F. Kammer, M. Löffler, and R. Silveira. Removing local extrema from imprecise terrains. *Comput. Geom.*, 45(7):334–349, 2012.

[10] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Int. J. Comput. Geometry Appl.*, 3(4):383–415, 1993.

[11] S. Har-Peled. On the expected complexity of random convex hulls. *CoRR*, abs/1111.5340, 2011.

[12] S. Har-Peled and B. Raichel. On the complexity of randomly weighted multiplicative voronoi diagrams. *Discret. Comput. Geom.*, 53(3):547–568, 2015.

[13] A. Jørgensen, M. Löffler, and J. M. Phillips. Geometric computations on indecisive points. In *Workshop on Algorithms and Data Structures (WADS)*, pages 536–547, 2011.

[14] V. Keikha, M. van de Kerkhof, M. J. van Kreveld, I. Kostitsyna, M. Löffler, F. Staals, J. Urhausen, J. L. Vermeulen, and L. Wiratma. Convex partial transversals of planar regions. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 52:1–52:12, 2018.

[15] N. Kumar, B. Raichel, S. Suri, and K. Verbeek. Most likely voronoi diagrams in higher dimensions. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 65 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[16] M. Löffler and J. Snoeyink. Delaunay triangulations of imprecise points in linear time after preprocessing. In *Proc. of 24th ACM Symp. on Comp. Geom. (SoCG)*, pages 298–304, 2008.

[17] M. Löffler and M. J. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. In *Workshop on Algorithms and Data Structures (WADS)*, pages 446–457, 2007.

[18] M. Löffler and M. J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[19] A. Mukhopadhyay, C. Kumar, E. Greene, and B. K. Bhattacharya. On intersecting a set of parallel line segments with a convex polygon of minimum area. *Inf. Process. Lett.*, 105(2):58–64, 2008.

[20] D. Rappaport. Minimum polygon transversals of line segments. *Int. J. Comput. Geometry Appl.*, 5(3):243–256, 1995.

[21] S. Suri, K. Verbeek, and H. Yildiz. On the most likely convex hull of uncertain points. In *European Symposium on Algorithms (ESA)*, pages 791–802, 2013.

A Vertical Line Segments

A.1 Missing Algorithm

The following algorithm is used as a subroutine in Algorithm 1, and is described in detail in the proof of Theorem 8.

Algorithm 2 Computes the set of valid positive or negative chain vertices

```
1:  function POSITIVE(i)
2:      P ← ∅
3:      s⁻_max ← s⁻_{i+1}
4:      for k ← i + 1 to n − 1 do
5:          if s⁺_k above line through s⁺_i and s⁻_max
                then P ← P ∪ {k}
6:          if s⁻_k above line through s⁺_i and s⁻_max
                then s⁻_max ← s⁻_k
7:      return P
8:  function NEGATIVE(j)
9:      N ← ∅
10:     s⁺_min ← s⁺_{j+1}
11:     for k ← j + 1 to n − 1 do
12:         if s⁻_k below line through s⁻_j and s⁺_min
                then N ← N ∪ {k}
13:         if s⁺_k below line through s⁻_j and s⁺_min
                then s⁺_min ← s⁺_k
14:     return N
```

A.2 Missing Proofs

Lemma 6. *Let P^+, P^- be a valid chain pair. Then $\mathcal{CH}(P^+ \cup P^-)$ intersects all segments in S.*

Proof. Note that P^+ and P^- both start on the same point on s_1 and end on the same point on s_n, and thus clearly $\mathcal{CH}(P^+ \cup P^-)$ intersects s_1 and s_n. So fix some segment $s = s_i$, for $1 < i < n$. From the lemma statement, there exists a point p from the chain P^+ which lies directly above s^- (p may be a vertex or an interior edge point). Similarly, define q as the point from P^- which lies directly below s^+. Thus we have $q.y \leq s^+.y$ and $s^-.y \leq p.y$. If $s^+.y \leq p.y$ then $q.y \leq s^+.y \leq p.y$ and hence $s^+ \in \overline{pq} = \mathcal{CH}(\{p, q\}) \subseteq \mathcal{CH}(P^+ \cup P^-)$. So assume otherwise, that $s^+.y > p.y$, which combined with our known inequality we have $s^+.y > p.y \geq s^-.y$. That is, p lies on the segment s, and hence $p \cap s = p \in \mathcal{CH}(P^+ \cup P^-)$. □

Lemma 9. *For a set S of n vertical segments, the optimal solution to Problem 1 where the top chain of the convex hull is not allowed to have interior vertices can be solved in $O(n^3)$ time.*

Proof. First consider the case when the bottom chain also has no interior vertices. Then we are looking for a single segment $\overline{lr}$ such that $l \in s_1$, $r \in s_2$, and which intersects all segments in S. Thus we are checking the feasibility of a linear program with $O(n)$ constraints in two variables, $l.y$ and $r.y$, which can be solved in $O(n^2)$ time by standard techniques (see [4]).

So now suppose the bottom chain has at least one interior vertex. First we precompute for every possible starting and ending index pair the minimal length negative subchain which could be in a valid chain pair. Specifically, for any pair of indices $1 < i \leq j < n$, let MINNEG(i, j) be the minimum number of vertices of a negative chain from s_i^- to s_j^- such that s_x^+ lies above the chain for all $i < x < j$. Observe that we can easily compute MINNEG(i, j) for all pairs $1 < i \leq j < n$ in $O(n^3)$ time using a similar but simpler dynamic programming approach as was done for MINCH(i, j) in Algorithm 1. Namely, the algorithm follows by the recursive relation MINNEG$(i, j) = 1 + \min_{k \in N}$ MINNEG(k, j) where N is the set of indices $i < k \leq j$, such that for all $i < x < k$, s_x^+ lies above the segment $\overline{s_i^- s_k^-}$. (N can be computed in linear time, similar to NEGATIVE(j) in Algorithm 2.)

By Corollary 7, we know the optimal solution is $2 + \min_{(i,j) \in V}$ MINNEG(i, j), where V is the set of all index pairs such that the minimal subchain computed by MINNEG(i, j) can be extended into a valid chain pair (such that the positive chain has no interior vertices). Specifically, V is the set of index pairs $1 < i \leq j < n$ where there exists points $l \in s_1$ and $r \in s_n$ such that 1) s_x^+ lies above $\overline{ls_i^-}$ for all $1 < x < i$, 2) s_x^+ lies above $\overline{s_j^- r}$ for all $j < x < n$, and 3) s_x^- lies below $\overline{lr}$ for all $1 < x < n$. Consider the first condition. Let $top = \min_{1<x<i} Int(s_x^+, s_i^-).y$, where $Int(s_x^+, s_i^-)$ denotes the point of intersection of the line supporting $\overline{s_x^+ s_i^-}$ with the vertical line supporting s_1. Then condition 1) is equivalent to requiring that $l.y \leq top$, and so this condition can be encoded by simply replacing the upper endpoint of s_1 with the point $(s_1^+.x, top)$ (if $top < s_1^-.y$ then $(i, j) \notin V$). Similarly, we can update the lower endpoint of s_n to handle condition 2) from above. Updating the endpoints in this manner takes $O(n)$ time for any given pair (i, j).

Thus all that remains is to handle condition 3). Here we require $\overline{lr}$ lies above s_x^- for all $1 < x < n$, where $l \in s_1$ and $r \in s_n$. Let E denote the set of all relevant endpoints, i.e. s_1^+, s_1^-, s_n^+, s_n^- and all s_x^- for $1 < x < n$. If such a segment $\overline{lr}$ exists, then we can translate it vertically downwards until it hits a point in E, and then rotate about that point until it hits a second endpoint in E, and it will still be a valid solution. Thus it suffices to limit our search to the set of all segments passing through two points in E. Now there are a few cases. First, suppose one of these two points is s_1^-. Consider the ray with base point s_1^- and pointing vertically up-

ward. Let s_k^- be the first point hit in the set of all s_x^- for $1 < x < n$, when rotating this ray clockwise. Clearly $\overline{lr}$ must pass above s_k^- and if it does then it passes above all s_x^- for $1 < x < n$. Thus in this case a valid $\overline{lr}$ exists if and only if the line supporting $\overline{s_1^- s_k^-}$ passes below s_n^+. Thus we can check all cases when one of the two points is s_1^- in $O(n)$ time, as this is how long it takes to compute s_k^-. A similar argument works for the cases when one of the two points is s_1^+, s_n^+, or s_n^-. So now suppose $\overline{lr}$ passes through two points s_g^- and s_h^-, such that $1 < g < h < n$. Observe that for $\overline{lr}$ to lie above s_x^- for all $1 < x < n$, this is equivalent to requiring $\overline{lr}$ to lie above the top chain of the convex hull of all such s_x^-. In other words, $\overline{s_g^- s_h^-}$ must define an edge of the top chain of the convex hull. There are only $O(n)$ such edges, all of which can be computed globally once in $O(n \log n)$ time (i.e. they do not need to be recomputed for each MinNeg(i, j)). For each such edge in constant time we can check whether the line supporting it intersects s_1 and s_n. Thus in $O(n)$ time (ignoring the global $O(n \log n)$ top chain computation) we can check all cases where $\overline{lr}$ passes through two points s_g^- and s_h^-, such that $1 < g < h < n$. So overall, for any pair (i, j) we can check in $O(n)$ time whether it lies in V, and thus by checking all pairs we can compute V in $O(n^3)$ time. □

A.3 Missing Remarks

Remark 12 *The case when s_1 is a vertical segment can be directly reduced to the single point case in Theorem 8, but the run time degrades. Imagine sliding a point p down the segment s_1. As we slide this point the behavior of* MinCH$(1, 1)$ *from Algorithm 1 only changes when either the set P or N change, and specifically as we slide p downwards the set P gets smaller and N larger. So fix an index k which initially is in P, and consider the moment when k leaves the set P. At this moment, for some $1 < x < k$, p must be the intersection of s_1 with the line supporting $\overline{s_x^- s_k^+}$, namely if p went any lower on s_1 then s_x^- would lie above $\overline{ps_k^+}$. A similar statement holds for changes in the set N. Thus consider the set of all $O(n^2)$ intersection points of the segment s_1 with lines supporting segments of the form $s_i^+ s_j^-$ for all pairs i, j. As all possible values for P, N are realizable by starting from some point in this canonical set of points, we can obtain the optimal solution to Problem 1 by calling the algorithm of Theorem 8 for each one of these points. As the running time of each call is $O(n^3)$, this would give an $O(n^5)$ time solution.*

Remark 13 *It is not hard to see that the approach in Section 2 also gives a polynomial time algorithm for Problem 1 when U is a set of axis-aligned rectangles that can be appropriately ordered. Specifically, suppose you are given the points l, r, t, b representing the leftmost, rightmost, topmost, and bottommost vertices of the optimal convex hull. Similar to Lemma 4, one can argue that there is an optimal solution where all the vertices on the top chain between l and t are realized at the upper left corner of their rectangle, and similar statements hold for the other corners. To use dynamic programming, however, we need to be given an ordering, such as the left to right order of the realizations of the rectangles. This would occur if, for example, the rectangles are separated by vertical lines, i.e. rectangles $R_1, \ldots R_n$ such that for any $i < j$, R_i lies entirely to the left of R_j. Note there are only a polynomial number of possibilities for l, r, t, b as their rectangles can be guessed, and there are only a polynomial number of canonical positions that to need be considered for their realization in each rectangle (similar to Remark 12). Thus this gives a polynomial time algorithm when we have such an ordering, though the constant would be high without similar optimizations as in the vertical segment case.*

B NP-Hardness for General Segments

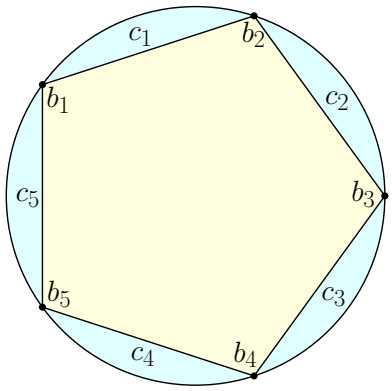


Figure B.1: Certain points $b_1, \ldots b_5$, and caps $c_1, \ldots, c_5$.

Sparse Convex Hull Coverage

Georgiy Klimenko* Benjamin Raichel* Gregory Van Buskirk*

Abstract

Given a set P of n data points and an integer k, a fundamental computational task is to find a smaller subset $Q \subseteq P$ of only k points which approximately preserves the geometry of P. Here we consider the problem of finding the subset Q of k points which best captures the convex hull of P, where our error measure is the sum of the distances of the points in P to the convex hull of Q. We generalize the problem to allow the set R that we must select Q from to differ from P, as well as to allow more general functions of the distances of the uncovered points of P, such as other norms or weighted distance functions.

We prove that approximating the convex hull in this manner in the plane can be solved by either a simple graph based or dynamic programming based algorithm in polynomial time. Complementing this result we show that in dimensions 3 and higher the problem is NP-hard. Moreover, we give an algorithm which in 3 dimensions selects $O(k \log(n/\varepsilon))$ points to get a solution whose error is at most $1+\varepsilon$ times the optimal k point error. This generalizes to $O(k^{\lfloor d/2 \rfloor} \log(n/\varepsilon))$ points for any constant dimension d.

1 Introduction

Given a point set $P \subset \mathbb{R}^d$, the convex hull of P, denoted $\mathcal{CH}(P)$, is a fundamental geometric structure, intuitively capturing the region covered by P. Here we consider the problem of covering P as best as possible by the convex hull of a subset of only k points from P, in effect sparsely approximating $\mathcal{CH}(P)$. This natural problem relates to the problem of approximating convex sets by polytopes, for which countless papers have been written (see the extensive survey [5]). Much of this previous work has focused on the objective of minimizing the maximum distance of an uncovered point from the hull of the selected points (i.e. Hausdorff distance), or approximating the volume in the case of smooth convex bodies. Here we instead study approximating the convex hull of a discrete point set under the objective of minimizing the sum of the distances of the uncovered points, an objective which when compared to the max objective is more robust to outliers as the error is no longer determined solely by the single furthest point. Our framework also allows for much more general cost functions of the distances, and in particular allows for any ℓ_p norm or weighted distance functions. We further generalize the problem such that the selected k points defining our hull are required to come from a set R that can differ from P, thus capturing scenarios where the covering objects differ from the covered ones. This is natural from a feature selection standpoint, where R represents a set of known possible features which we wish to represent a set of observed objects P. For such problems the convex hull is a particularly relevant structure as it represents the set of all weighted averages of the selected points. Moreover, the Carathéodory theorem states that any point in the convex hull of the chosen subset can be represented as a convex combination of $d+1$ of the chosen points, yielding a sparse representation in low dimensions. (In higher dimensions one can use the approximate Carathéodory theorem [2].)

More generally, given a set $P \subset \mathbb{R}^d$ of n points, finding a smaller set of only k points which approximately captures the geometry of P under some measure is a ubiquitous computational task. Two standard such problems of interest are k-clustering and subspace fitting. In k-clustering the objective is to select a subset of k center points so as to minimize some norm of the vector of distances from each point in P to its nearest center. For example, k-means seeks to minimize the ℓ_2 norm [1], where it is known that even planar k-means is NP-hard [11]. For subspace fitting the objective is to select the k-dimensional subspace minimizing some norm of the distances to the linear subspace, e.g. the solution under the ℓ_2 norm is known to be the top k singular vectors when viewing P as a matrix. If one restricts the selected k points to come from P, then the clustering and subspace fitting problems become the standard discrete k-clustering and CUR-decomposition [4] problems.

Our problem of approximating the convex hull can be viewed as naturally lying between clustering and subspace fitting, when restricting the selected subset to come from a set R. Specifically, viewing the selected subset of k points $Q \subseteq R$ as a basis, the problems are defined by how we allow each point in P to be represented by Q. In subspace fitting, any linear combination is allowed, in convex hull coverage only convex combinations are allowed (i.e. non-negative and summing to 1), and in clustering not only are the combinations con-

*Department of Computer Science, University of Texas at Dallas, {gik140030, benjamin.raichel, greg.vanbuskirk}@utdallas.edu. Work on this paper was partially supported by a NSF CAREER Award 1750780.

vex but are all zero except for a single 1 (i.e. the nearest center). That is, one can define an entire spectrum of problems based on how one restricts reconstruction from the basis, and convex hull coverage is a natural set point on this spectrum. In this sense, other standard problems such as non-negative matrix factorization (NMF), which is known to NP-hard [15], can be seen as another set point on this spectrum. (NMF typically restricts the basis to non-negative vectors, though restricting to input points is also commonly studied [10].)

Another related topic is coresets, which are small subsets of the input which can be used as a proxy for the full set. There are numerous coresets results (see chapter 48 in [13]). Relevant to the current paper, it is known that for any point set P contained in the unit ball,[1] there is a subset $S \subseteq P$ of $O(1/\varepsilon^{(d-1)/2})$ points such that all of P is with distance ε from $\mathcal{CH}(S)$. Worst case point sets require such an exponential dependence on d, and thus [3] considered coresets whose size is measured relative to the given instance, showing that if some k points achieves ε error, then a greedy algorithm selecting $O(k/\varepsilon^{2/3})$ points achieves $O(\varepsilon^{1/3})$ error. This result was later extended by [14] to get analogous results for approximating the conic hull, which consists of all non-negative combinations, and thus relates to NMF.

Our Contribution. For point sets $R, P \subset \mathbb{R}^d$ of m and n points, respectively, we initiate the rigorous study of the convex hull coverage problem, where the goal is to find a subset $Q \subseteq R$ of k points minimizing the sum of distances from the points in P to their projection onto the convex hull of Q, that is $\sum_{p\in P} ||p - \mathcal{CH}(Q)||$. Furthermore, we generalize the problem to allow any cost function of the form $\sum_{p\in P} g_p(||p-\mathcal{CH}(Q)||)$, where each g_p can be any monotonically increasing real valued function such that $g_p(\alpha) = 0$ if and only if $\alpha = 0$. Thus we can model for example weighted sums or other ℓ_p norms of the distances of the points in P to the hull (by taking the pth power of the norm).

We prove that convex hull coverage can be solved exactly in the plane in $O(m^3k + m^2n + mn\log(n))$ time via dynamic programming. Interestingly, for the special case when $P = R$, we can show that by carefully assigning weights the problem nicely reduces to the problem of finding a minimum cost k length cycle in a directed graph. This yields a simpler graph based algorithm with $O(n^3 \log k)$ running time. To complement our results in the plane, we argue that the convex hull coverage problem is NP-hard for $d \geq 3$, even when restricting our objective to the sum of distances (i.e. the g_p are all the identity function). Furthermore, we argue that even if one restricts to instances where $P = R$, the problem remains NP-hard for $d \geq 4$. Finally, we argue that a geometric set cover based algorithm yields an approximation in constant dimensions for the sum of distances. Namely, for $d = 3$ greedily selecting $O(k \log(n/\varepsilon))$ points in an appropriate way gives a solution whose error is at most $1 + \varepsilon$ times the optimal k point error. This generalizes to $O(k^{\lfloor d/2 \rfloor} \log(n/\varepsilon))$ points for any constant dimension d.

One of the main challenges of convex hull coverage for $d \geq 3$ is that it lacks certain independence properties of related problems. For example, in k-clustering, the cluster centers partition the points based on their nearest center, whereas the projection of a point onto the convex hull is determined by several hull vertices. For subspace approximation under the Frobenius norm there is independence among the dimensions, in the sense that the kth singular vector is determined by finding the optimum vector in the orthogonal subspace of the first $k-1$ singular vectors. Note also that previous coreset results focused on the max measure, where a given error ε represents a precise constraint that all points must satisfy. On the other hand, for our sum measure, an error ε represents a budget that the algorithm must now decide how to allocate amongst the various points.

[1] Any point set can be scaled to lie in the unit ball, effectively meaning ε is measured relative to the diameter before scaling, which is in some sense necessary. Via an affine transformation, one can ague such coresets exist for directional width where error is relative to the diameter in each direction, see [9].

2 Preliminaries

Given a point set X in $\mathbb{R}^d$, let $\mathcal{CH}(X)$ denote its convex hull. For two points $x, y \in \mathbb{R}^d$, let xy denote their line segment, that is $xy = \mathcal{CH}(\{x, y\})$. Throughout, given points $x, y \in \mathbb{R}^d$, $||x - y||$ denotes their Euclidean distance. Given two compact sets $X, Y \subset \mathbb{R}^d$, $||X - Y|| = \min_{x\in X, y\in Y} ||x - y||$ denotes their distance. For a single point x we write $||x - Y|| = ||\{x\} - Y||$.

Definition 1 *Let $P \subset \mathbb{R}^d$ be a set of n points, where for each point $x \in P$, there is an associated monotonically increasing real valued function g_x such that $g_x(\alpha) = 0$ if and only if $\alpha = 0$. Then we call any function of the form $f(Q, P') = \sum_{x\in P'} g_x(||x-\mathcal{CH}(Q)||)$, where $Q \subset \mathbb{R}^d$ and $P' \subseteq P$, a* hull coverage function. *We let $\mathcal{F}_P$ denote the set of all such functions.*

In the above definition we assume the g_x functions can be evaluated in constant time. The following is the main problem studied in this paper.

Problem 2 *Given a set $P \subset \mathbb{R}^d$ of n points, a set $R \subset \mathbb{R}^d$ of m points, and a function $f \in \mathcal{F}_P$, select a subset $Q \subseteq R$ of at most k points which minimizes $f(Q, P)$. That is, $Q = \arg\min_{Q\subseteq R, |Q|\leq k} f(Q, P)$.*

3 Exact Computation in the Plane

In this section we give polynomial time algorithms for Problem 2 when $d = 2$. First, we give a simple graph

based algorithm for the special case when $P = R$, followed by a slightly more involved dynamic programming algorithm for the general case.

3.1 A graph algorithm for a simpler case

In this section we argue that by assuming $P = R$, one can solve Problem 2 in the plane by converting it into a corresponding graph problem. Specifically, construct a weighted and fully connected directed graph $G_P = (V, E)$ where $V = P$. Given an order pair of points p, q, let $P_{p,q}$ denote the subset of P in the closed halfspace whose boundary is the line through p and q and lies to the left of the ray from p to q. Then we define the weight of the directed edge (p, q) to be $w(p, q) = f(\{p, q\}, P_{p,q})$. For a cycle of vertices $C = \{p_1, \ldots, p_k\}$, let $w(C)$ denote the sum of the weights of the directed edges around the cycle. Throughout, we only consider non-trivial cycles, that is cycles must have at least two vertices.

For a set of points Q, let $\mathcal{CH}_L(Q)$ denote the clockwise list of vertices on the boundary of $\mathcal{CH}(Q)$. Observe that any subset $Q \subseteq P$ corresponds to the cycle $\mathcal{CH}_L(Q)$ in G_P. Moreover, any cycle C correspond to the convex hull $\mathcal{CH}(C)$.

Lemma 3 *Consider an instance P, R, f, k of Problem 2 in the plane where $P = R$. Let C be any cycle in G_P, and let Q be an optimal solution. Then,*
1) $w(C) \geq f(C, P)$, 2) $w(\mathcal{CH}_L(Q)) = f(Q, P)$.

Proof. First, observe that $w(C)$ and $f(C, P)$ can be decomposed into the contribution of each point.

$$f(C,P) = \sum_{p \in P} g_p(||p - \mathcal{CH}(C)||) \qquad \text{and}$$

$$w(C) = \sum_{(a,b)\in C} f(\{a,b\}, P_{a,b}) = \sum_{p \in P} \sum_{\substack{(a,b)\in C \\ \text{s.t. } p \in P_{a,b}}} g_p(||p - ab||).$$

To prove the first part of the lemma, we thus argue that for any $p \in P$, its contribution to $w(C)$ is at least as large as its contribution to $f(C, P)$. Assume $p \notin \mathcal{CH}(C)$, since otherwise it does not contribute to $f(C, P)$. It suffices to argue there exists an edge $(a, b) \in C$, such that $p \in P_{a,b}$, since $||p-ab|| \geq ||p-\mathcal{CH}(C)||$ and g_p is a monotonically increasing function. So assume otherwise that there is some point $p \in P$ such that p lies to the right of all edges in C. Create a line ℓ that passes through p and any interior point of any edge $(a, b) \in C$, but does not pass through any point in R. ℓ splits the plane into two halfspaces. As p is to the right of any edge and is outside the convex hull of the points, all edges intersecting ℓ have to begin at the same halfspace, and end at the other halfspace. This implies C is not a cycle, which is a contradiction.

To prove the second part of the lemma for an optimal solution Q, we argue that for any $p \in P$, its contribution to $f(Q, P)$ is equal to its contribution to $w(\mathcal{CH}_L(Q))$. If $p \in \mathcal{CH}(Q)$ then it lies to the right of all edges in $\mathcal{CH}_L(Q)$, and so its contributions to both $w(\mathcal{CH}_L(Q))$ and $f(Q, P)$ are zero. So consider a point $p \notin \mathcal{CH}(Q)$. Let ab be the closest edge of $\mathcal{CH}(Q)$ (where b follows a in clockwise order). Note that $||p-\mathcal{CH}(Q)|| = ||p-ab||$ and $p \in P_{ab}$, and thus the contributions of p to $f(Q, P)$ and $w(\mathcal{CH}_L(Q))$ are equal if and only if p lies to right of all other edges in $\mathcal{CH}_L(Q)$, as otherwise p has a positive contribution to another edge since by definition $g_p(\alpha) > 0$ for $\alpha > 0$. So suppose otherwise, that p lies to the left of some other edge cd (note it may be that $b = c$). Thus p is in the intersection of the halfspace to the left of the line from a through b and to the left of the line from c through d, see Figure 3.1. This implies that $b, c \in \mathcal{CH}(\{a, d, p\})$. So let $Q' = Q \cup \{p\} \setminus \{b, c\}$, then $\mathcal{CH}(Q) \subset \mathcal{CH}(Q')$. This implies $f(Q', P) < f(Q, P)$ as Q' contains p but Q does not, which is a contradiction with Q being an optimal solution as $|Q'| \leq |Q|$. (Note that assuming $P = R$ was used to ensure that Q' was a possible solution.) □

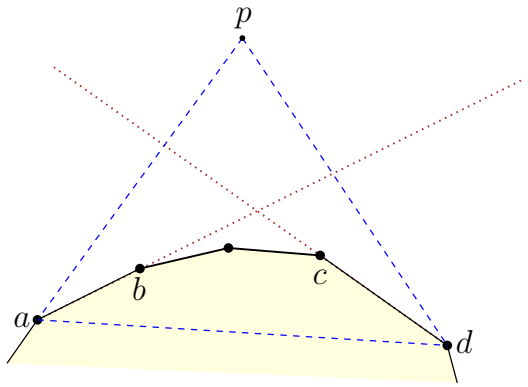


Figure 3.1: $b, c \in \mathcal{CH}(\{a, d, p\})$ when $p \in P_{ab}$ and $p \in P_{cd}$.

Theorem 4 *Given an instance P, R, f, k of Problem 2 in the plane where $P = R$, it can can be solved in $O(n^3 \log k)$ time, where $n = |P| = |R|$.*

Proof. Let C be a minimum cost cycle in G_R subject to having at most k vertices. The claim is that the set of vertices in C is an optimal solution to Problem 2, that is, $f(C, P) = \min_{X \subseteq R, |X| \leq k} f(X, P)$. By part 1) of Lemma 3, $w(C) \geq f(C, P)$, and thus if C is not optimal, then the optimal solution must have cost strictly less than $w(C)$. However, by part 2) of Lemma 3, the optimal solution corresponds to a cycle in G_R with the same cost, which contradicts C being minimum cost.

Now we analyze the running time. Computing G_R takes $O(n^3)$ time as there are $O(n^2)$ edges, and computing the weight of each edge takes $O(n)$ time, as it is a sum of at most n constant time computable functions. To compute the minimum cost cycle with $\leq k$

edges, it suffices to compute the all pairs shortest path distances for paths with $\leq k-1$ edges, since afterwards in $O(n^2)$ time we can add the final edge of each cycle. It is known that for a graph with n vertices the all pairs shortest path distances for paths with $\leq k-1$ edges can be computed in $O(n^3 \log k)$ time, see for example the matrix multiplication algorithm in [7]. Thus, the overall running time is $O(n^3 \log k)$. □

3.2 Dynamic programming for the general case

We now argue that when P is allowed to differ from R we can still compute the optimal solution in the plane in polynomial time by using a slightly more involved and slightly slower dynamic program.

Let $V = \{v_1, \ldots, v_k\} \subseteq R$ be the vertices of some convex hull of points from R, labeled in clockwise order, where v_1 is the vertex of V with smallest y-coordinate. Consider our cost function $f(V, P) = \sum_{x \in P} g_x(||x - \mathcal{CH}(V)||)$. Any point $x \in \mathcal{CH}(V)$ contributes zero to f, as we required $g_x(0) = 0$. So consider any point $x \in P$ lying outside of $\mathcal{CH}(V)$. The projection of x onto $\mathcal{CH}(V)$ is either a vertex v_i or a point on the interior of an edge $v_{i-1}v_i$, for some i. Thus the edges and vertices of the hull define a partition of points in P which lie outside the hull, which we now formally describe.

Consider the ray with base point v_{i-1} and directed from v_{i-1} towards v_i. Define $r_l(v_{i-1}, v_i)$ to be the rotation of this ray by $\pi/2$ to the left, that is the ray with base point v_{i-1} and direction $(v_{i-1}.y - v_i.y, v_i.x - v_{i-1}.x)$. Define $r_r(v_{i-1}, v_i)$ to be ray with the same direction, but with base point v_i. Then $slab(v_{i-1}, v_i)$ is defined as the region of the plane interior to and bounded by the edge $v_{i-1}v_i$ and (between) the rays $r_l(v_{i-1}, v_i)$ and $r_r(v_{i-1}, v_i)$. See Figure 3.2. Define $cone(v_{i-1}, v_i, v_{i+1})$ as the closed region bounded $r_r(v_{i-1}, v_i)$ and $r_l(v_i, v_{i+1})$, again see Figure 3.2. In other words, $slab(v_{i-1}, v_i)$ and $cone(v_{i-1}, v_i, v_{i+1})$ are the subsets of points in the plane outside of $\mathcal{CH}(V)$ whose projection onto $\mathcal{CH}(V)$ lies on the interior of $v_{i-1}v_i$ or on the vertex v_i, respectively. In particular, for a point set P, define

$$sum_{slab}(v_{i-1}, v_i) = f(\{v_{i-1}, v_i\}, P \cap slab(v_{i-1}, v_i)) = \sum_{p \in slab(v_{i-1}, v_i)} g_p(||p - \mathcal{CH}(\{v_{i-1}, v_i\})||)$$

$$sum_{cone}(v_{i-1}, v_i, v_{i+1}) = f(\{v_i\}, P \cap cone(v_{i-1}, v_i, v_{i+1})) = \sum_{p \in cone(v_{i-1}, v_i, v_{i+1})} g_p(||p - v_i||)$$

Observe that $sum_{slab}(v_{i-1}, v_i)$ only depends on v_{i-1} and v_i and $sum_{cone}(v_{i-1}, v_i, v_{i+1})$ only depends on v_{i-1}, v_i, and v_{i+1}. In particular, these quantities are respectively defined for any pair or triple of points in R, and for now assume they have all been precomputed.

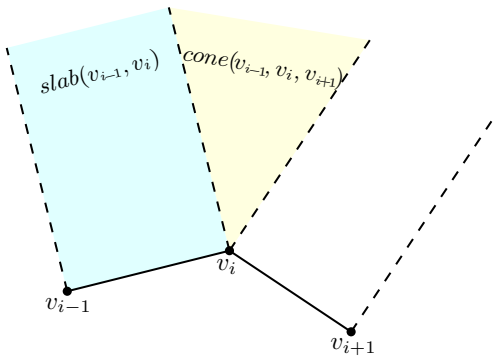


Figure 3.2: Three consecutive vertices on the hull, and the corresponding defined slabs and cone.

By the discussion above, for ordered vertices V of a convex hull we can rewrite our cost function as

$$f(V, P) = \sum_{x \in P} g_x(||x - \mathcal{CH}(V)||) = \sum_{i=1}^{k} (sum_{cone}(v_{i-1}, v_i, v_{i+1}) + sum_{slab}(v_i, v_{i+1})), \tag{3.1}$$

where indices are mod k, i.e. $v_0 = v_k$ and $v_{k+1} = v_1$. This equation suggests a natural recursive strategy to minimize $f(V, P)$ (over choices of V) by guessing the vertices of V in clockwise order.

First, at the cost of an additional linear factor in the running time, we guess the point with the smallest y-coordinate from the optimal hull.[2] We call this the *starting point* and denote it by s (i.e. $v_1 = s$). Let R_s be the subset of points in R whose y-coordinate is greater than that of s. As we assumed s is the lowest point in the optimal solution, we can disregard points in $R \setminus R_s$. Next, we sort all other points in R_s clockwise radially around s (i.e. from the negative x axis clockwise about s to the positive x axis) and process points in this order.

One issue we must deal with first is that in Equation 3.1, $sum_{cone}(v_k, s, v_2)$ depends both on the choice of v_k and v_2. To break this cyclic behavior we cut the cone for s in two. So cast a ray in the negative y-direction from s and call it r_s, and observe that as s is the lowest vertex r_s must lie in the cone for s. We cut the cone for s along r_s and assign each piece to its adjacent slab. Specifically, suppose we set $v_2 = u$ for some $u \in R_s$. Then define $sum_{start}(s, u)$ as the union the region $sum_{slab}(s, u)$ with the cone lying between the rays r_s and $r_l(s, u)$ (including r_s). Similarly, if we set $v_k = w$, then define $sum_{end}(w, s)$ as the union of the region $sum_{slab}(w, s)$ with the cone lying between the rays

[2] We can assume all points have distinct y-coordinates, by applying a small random rotation, which does not affect f.

$r_r(w,s)$ and r_s (excluding r_s). Then we have,

$$f(V,P) = sum_{start}(s,v_2)+ \sum_{i=2}^{k-1}(sum_{cone}(v_{i-1},v_i,v_{i+1}) + sum_{slab}(v_i,v_{i+1})) + (sum_{cone}(v_{k-1},v_k,s) + sum_{end}(v_k,s)). \quad (3.2)$$

Given the above equation breaking the cost function into a linear ordered set of cones and slabs, it is relatively straightforward to compute the optimal solution using dynamic programming. Due to space, the pseudocode and proof have been moved to Appendix A. We remark that achieving the specific running time of the following summarizing theorem though is non-trivial. In particular, a roughly $O(m)$ factor is saved over the naive time bound by using sweeping both to batch dynamic programming table entries together and to implicitly precompute the sum_{cone} values.

Theorem 5 *Given an instance P, R, f, k of Problem 2 in the plane, it can be solved in $O(m^3k + m^2n + mn\log(n))$ time, where $n = |P|$ and $m = |R|$.*

4 Hardness in Higher Dimensions

A convex polytope $T = (V, E)$ in $\mathbb{R}^3$, will be defined as a graph where the vertices V are a set of points in convex position in $\mathbb{R}^3$, and the edges E are the edges of $\mathcal{CH}(V)$. [8] proved the following variant of vertex cover is NP-hard.

Problem 6 (Polytope Vertex Cover) *Given a convex polytope $T = (V, E)$ in $\mathbb{R}^3$ and an integer k, is there a subset $U \subseteq V$ of k vertices such that each edge in E is incident to a vertex in U?*

The following is the decision version of our main problem, Problem 2.

Problem 7 *Given a set $P \subset \mathbb{R}^d$ of n points, a set $R \subset \mathbb{R}^d$ of m points, a function $f \in \mathcal{F}_P$, and a parameter ε, is there a subset $Q \subseteq R$ of at most k points such that $f(Q,P) \leq \varepsilon$.*

We now show Problem 7 is NP-hard for $d \geq 3$, where $f(Q,P) = \sum_{x\in P} g_x(||x - \mathcal{CH}(Q)||)$ is a natural and simple function. Namely, we set $g_x(||x - \mathcal{CH}(Q)||) = ||x - \mathcal{CH}(Q)||$ for all x. We denote this sum of distances function as $\mathsf{sd}(Q,P) = \sum_{x\in P} ||x - \mathcal{CH}(Q)||$.

Theorem 8 *Problem 7 is NP-hard for $d \geq 3$, $f = \mathsf{sd}$.*

Proof. We give a polynomial time reduction from Problem 6. Let $T = (V, E)$ and k be an instance of Problem 6. We first define several quantities based on T. For any edge $e \in E$, define a vector $u_e = (n_1 + n_2)/2$, where n_1 and n_2 are the normals of the planes of the two faces adjacent to e. For any edge $e = (v_1, v_2)$, consider the plane z_e containing e and with normal u_e. Let h_e be the distance from z_e to the convex hull of V after removing the endpoints of e, i.e. $h_e = ||z_e - \mathcal{CH}(V \setminus \{v_1, v_2\})||$, and let $h = \min_{e\in E} h_e$. (Note h is non-zero as V is in convex position.) Finally, let l_e be the length of the edge e, and let $l = \max_{e\in E} l_e$.

We construct our instance of Problem 7 as follows. We use the same value of k, and set $R = V$. P will contain one point for each edge $e \in E$, denoted p_e. We place p_e outside $\mathcal{CH}(V)$ at a distance x in the direction of u_e from the midpoint of e, where x is a value to be determined shortly. Finally, we set $\varepsilon = n\sqrt{x^2 + (l/2)^2}$, and recall $f(Q,P) = \mathsf{sd}(Q,P) = \sum_{p\in P} ||p - \mathcal{CH}(Q)||$.

Observe that for any edge $e \in E$, if at least one of its endpoints is selected, then the distance from p_e to the hull of the selected vertices is at most $\sqrt{x^2 + (l_e/2)^2} \leq \sqrt{x^2 + (l/2)^2}$. Thus if $U \subseteq V$ is a vertex cover of V, then $\mathsf{sd}(U,P) \leq n\sqrt{x^2 + (l/2)^2} = \varepsilon$. On the other hand if U is not a vertex cover, then there is an edge e for which neither endpoint is selected, in which case the distance from p_e to the hull of the selected vertices is at least $x + h$. Thus the total distance of all points to the hull is at least $(n-1)x + (x + h) = nx + h$, as by construction for any $e' \in E$ we have $||p_{e'} - \mathcal{CH}(R)|| = x$. Thus if we select x such that $nx + h > \varepsilon$, then U is vertex cover if and only if $\mathsf{sd}(U,P) \leq \varepsilon$. To ensure this inequality holds, set $x = \frac{l^2 n}{8h}$. Then we have

$$\varepsilon = n \cdot \sqrt{x^2 + \frac{l^2}{4}} = n \cdot \sqrt{\left(\frac{l^2 n}{8h}\right)^2 + \frac{l^2}{4}} < n \cdot \sqrt{\left(\frac{l^2 n}{8h} + \frac{h}{n}\right)^2} = n \cdot \frac{l^2 n}{8h} + h = nx + h.$$

□

By lifting to $\mathbb{R}^4$ we can argue that the problem remains NP-hard for the restricted variant where $P = R$, i.e. the case considered in Section 3.1. The proof is more technically challenging, though at a high level uses a similar approach and thus has been moved to Appendix B for space.

Theorem 9 *Problem 7 is NP-hard for $d \geq 4$, $f = \mathsf{sd}$, and $P = R$.*

5 Approximation in Higher Constant Dimensions

Given the hardness of our problem when $d \geq 3$, it is natural to consider approximations. For the Set Cover problem, it is well known that if k sets cover the ground set, then the greedy algorithm covers the ground set with $O(k \log n)$ sets. Our hull problem is also a coverage problem, though it is more challenging as the points

in P are not covered by the individual points we select but rather convex combinations of them. Despite this, we argue a similar greedy approach works, though it depends on the number of facets of the convex hull of the optimal k point solution. In 3d the number of facets is $O(k)$, yielding a $(1+\varepsilon)$ approximation to the error with only $O(k\log(n/\varepsilon))$ points, similar to Set Cover. In higher constant dimensions, however, the worst case facet complexity is $O(k^{\lfloor d/2 \rfloor})$. On real world inputs the complexity may be significantly lower (see [12] for the facet complexity of randomly sampled points), thus our analysis suggests that greedily selecting roughly a logarithmic factor more points may be a reasonable heuristic in practice for small constant dimensions.

In this section we assume P and R are contained in the unit ball, which as remarked in the introduction is equivalent to measuring the error relative to the diameter, as is standard.

Previously we considered the sum of distances function $\mathsf{sd}(Q,P) = \sum_{x\in P} ||x - \mathcal{CH}(Q)||$. Similarly, we can define the maximum distance function $\mathsf{md}(Q,P) = \max_{x\in P} ||x - \mathcal{CH}(Q)||$. We have the following corresponding optimization problem, considered in [3].

Problem 10 *Given a set $P \subset \mathbb{R}^d$ of n points and a set $R \subset \mathbb{R}^d$ of m points, select a subset $Q \subseteq R$ of at most k points which minimizes $\mathsf{md}(Q,P)$. That is, $Q = \arg\min_{Q\subseteq R, |Q|\le k} \mathsf{md}(Q,P)$.*

For an instance $P, R \subset \mathbb{R}^d$ and k of Problem 10, define

$$\mathsf{opt}_{\mathsf{md}} := \mathsf{opt}_{\mathsf{md}}(P,R,k) = \arg\min_{Q\subseteq R,\,|Q|\le k} \mathsf{md}(Q,P), \quad \text{and} \quad \overline{\mathsf{opt}_{\mathsf{md}}} = \mathsf{md}(\mathsf{opt}_{\mathsf{md}},P).$$

Similarly define

$$\mathsf{opt}_{\mathsf{sd}} := \mathsf{opt}_{\mathsf{sd}}(P,R,k) = \arg\min_{Q\subseteq R,\,|Q|\le k} \mathsf{sd}(Q,P), \quad \text{and} \quad \overline{\mathsf{opt}_{\mathsf{sd}}} = \mathsf{sd}(\mathsf{opt}_{\mathsf{sd}},P).$$

Lemma 11 ([3]) *Let $P, R \subset \mathbb{R}^d$ and k be an instance of Problem 10, where d is a constant. Then in polynomial time one can compute a set Q_0 of $O(k\log k)$ points such that $\mathsf{md}(Q_0,P) \le \overline{\mathsf{opt}_{\mathsf{md}}}(P,R,k)$.*

Let Q_0 be the set described in the above lemma. Observe that

$$\begin{aligned}\frac{\mathsf{sd}(Q_0,P)}{n} &\le \mathsf{md}(Q_0,P) \le \overline{\mathsf{opt}_{\mathsf{md}}} \\ &= \max_{p\in P} ||p - \mathcal{CH}(\mathsf{opt}_{\mathsf{md}})|| \le \max_{p\in P} ||p - \mathcal{CH}(\mathsf{opt}_{\mathsf{sd}})|| \\ &\le \sum_{p\in P} ||p - \mathcal{CH}(\mathsf{opt}_{\mathsf{sd}})|| = \overline{\mathsf{opt}_{\mathsf{sd}}},\end{aligned}$$

that is Q_0 achieves an n-approximation to the optimal sum distance cost $\overline{\mathsf{opt}_{\mathsf{sd}}}$.

For any subset $Q \subseteq R$, let $Z(Q,P) = \mathsf{sd}(Q,P) - \mathsf{sd}(\mathsf{opt}_{\mathsf{sd}},P) = \mathsf{sd}(Q,P) - \overline{\mathsf{opt}_{\mathsf{sd}}}$. For convenience $Z(Q)$ will denote $Z(Q,P)$ when P is the full point set. The proof of the following helper lemma is in Appendix C.

Lemma 12 *Given an instance $P, R \subset \mathbb{R}^d$ and k of Problem 2, where $f = \mathsf{sd}$ and d is a constant, for any subset $Q \subseteq R$ such that $Z(Q) \ge 0$, there exists a d-simplex Δ such that $Z(Q\cup\Delta) \le \left(1 - \frac{1}{c\cdot k^{\lfloor d/2\rfloor}}\right) Z(Q)$, where c is a constant.*

We remark that the running time of Lemma 11 from [3] depends exponentially on d, and thus the same is true for the following theorem which makes use of it.

Theorem 13 *Given an instance $P, R \subset \mathbb{R}^d$ and k of Problem 2, where $f = \mathsf{sd}$ and d is a constant, in polynomial time one can compute a set $Q \subseteq R$ of $O(k^{\lfloor d/2\rfloor}\log(n/\varepsilon))$ points such that $\mathsf{sd}(Q,P) \le (1+\varepsilon)\cdot \overline{\mathsf{opt}_{\mathsf{sd}}}(P,R,k)$.*

Proof. Use Lemma 11 to compute a set $Q_0 \subseteq R$ of $O(k\log k)$ points such that $\mathsf{sd}(Q_0,P) \le n\cdot\overline{\mathsf{opt}_{\mathsf{sd}}}(P,R,k)$. We will iteratively add subsets of $d+1$ points to Q_i for $i = \{0,1,\ldots,m-1\}$ where m is the total number of iterations. Let $A_i := \arg\min_{\Delta\subseteq R, |\Delta| = d+1} \mathsf{sd}(Q_i\cup\Delta, P)$ that is, A_i is the d-simplex whose addition to the current hull minimizes the sum of distances. In the ith iteration we add A_i to Q_i to obtain $Q_{i+1} := Q_i \cup A_i$.

Recall that $Z(Q_m) = \mathsf{sd}(Q_m,P) - \overline{\mathsf{opt}_{\mathsf{sd}}}$. Thus if $Z(Q_m) \le \varepsilon\cdot\overline{\mathsf{opt}_{\mathsf{sd}}}$ then $\mathsf{sd}(Q_m,P) \le (1+\varepsilon)\overline{\mathsf{opt}_{\mathsf{sd}}}$ as desired. If at any iteration $Z(Q_i) \le 0$, then $Z(Q_m) \le 0 \le \varepsilon\cdot\overline{\mathsf{opt}_{\mathsf{sd}}}$, since adding more points in later iterations can only further decrease the error. So assume that $Z(Q_i) > 0$, then by lemma Lemma 12, there exists a simplex Δ such that $Z(Q_i\cup\Delta) \le \left(1 - \frac{1}{c\cdot k^{\lfloor d/2\rfloor}}\right) Z(Q_i)$. Note that since $Z(Q_i\cup\Delta) = \mathsf{sd}(Q_i\cup\Delta,P) - \overline{\mathsf{opt}_{\mathsf{sd}}}$, we have $Z(Q_{i+1}) = Z(Q_i\cup A_i) \le Z(Q_i\cup\Delta)$ since we chose A_i to minimize $\mathsf{sd}(Q_i\cup A_i,P)$ and $\overline{\mathsf{opt}_{\mathsf{sd}}}$ is fixed. Thus we have $Z(Q_{i+1}) \le \left(1 - \frac{1}{c\cdot k^{\lfloor d/2\rfloor}}\right) Z(Q_i)$, and inductively

$$\begin{aligned}Z(Q_m) &\le \left(1 - \frac{1}{c\cdot k^{\lfloor d/2\rfloor}}\right)^m Z(Q_0) \\ &\le \left(1 - \frac{1}{c\cdot k^{\lfloor d/2\rfloor}}\right)^m n\cdot\overline{\mathsf{opt}_{\mathsf{sd}}},\end{aligned}$$

where the second inequality follows as $\mathsf{sd}(Q_0,P) \le n\cdot\overline{\mathsf{opt}_{\mathsf{sd}}}$. Thus if we select m such that $\left(1 - \frac{1}{c\cdot k^{\lfloor d/2\rfloor}}\right)^m \le (\varepsilon/n)$, then $Z(Q_m) \le \varepsilon\cdot\overline{\mathsf{opt}_{\mathsf{sd}}}$ as desired. Note that $(1 - \frac{1}{ck^{\lfloor d/2\rfloor}})^m \le exp(m/ck^{\lfloor d/2\rfloor})$ and rearranging the equation $exp(m/ck^{\lfloor d/2\rfloor}) = \varepsilon/n$ gives $m = ck^{\lfloor d/2\rfloor}\log(n/\varepsilon)$. As we are adding $d+1$ points in each round, and d is a constant, we thus get $O(k^{\lfloor d/2\rfloor}\log(n/\varepsilon))$ points in total. ☐

Corollary 14 *Given an instance $P, R \subset \mathbb{R}^3$ and k of Problem 2, where $f = \mathsf{sd}$, in polynomial time one can compute a set $Q \subseteq R$ of $O(k\log(n/\varepsilon))$ points such that $\mathsf{sd}(Q,P) \le (1+\varepsilon)\cdot\overline{\mathsf{opt}_{\mathsf{sd}}}(P,R,k)$.*

References

[1] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007.

[2] S. Barman. Approximating nash equilibria and dense bipartite subgraphs via an approximate version of caratheodory's theorem. pages 361–369. ACM, 2015.

[3] A. Blum, S. Har-Peled, and B. Raichel. Sparse approximation via generating point sets. *ACM Trans. Algorithms*, 15(3):32:1–32:16, 2019.

[4] C. Boutsidis and D. Woodruff. Optimal CUR matrix decompositions. *SIAM J. Comput.*, 46(2):543–589, 2017.

[5] E. Bronstein. Approximation of convex sets by polytopes. *Journal of Mathematical Sciences*, 153(6):727–762, 2008.

[6] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[8] G. Das and M. Goodrich. On the complexity of optimization problems for 3-dimensional convex polyhedra and decision trees. *Comput. Geom.*, 8:123–137, 1997.

[9] S. Har-peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

[10] A. Kumar, V. Sindhwani, and P. Kambadur. Fast conical hull algorithms for near-separable nonnegative matrix factorization. In *Int. Conf. on Machine Learning*, pages 28:231–239, 2013.

[11] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is np-hard. *Theor. Comput. Sci.*, 442:13–21, 2012.

[12] M. Reitzner. The combinatorial structure of random polytopes. *Advances in Mathematics*, 191(1):178 – 208, 2005.

[13] C. Tóth, J. O'Rourke, and J. Goodman. *Handbook of Discrete and Computational Geometry*. Discrete Mathematics and its Applications. CRC Press, 2017.

[14] G. Van Buskirk, B. Raichel, and N. Ruozzi. Sparse approximate conic hulls. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2534–2544, 2017.

[15] S. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, 2010.

A Proofs from Section 3.2

We now argue that the recursive algorithm shown in Algorithm 1, minimizes Equation 3.2 over all $V \subseteq R$, such that $V = \{v_1 = s, v_2, \ldots, v_k\}$ are the ordered vertices of a convex hull with lowest point s. This algorithm makes use of the function $right(u, v, w)$ which returns true if the ordered triple (u, v, w) represents a right turn and returns false otherwise. The following simple helper lemma ensures that we do not need to check for a right turn at s (i.e. where we split the problem), as long as we check everywhere else.

Lemma 15 *Let $V = \{v_1, v_2, \ldots, v_k\}$ be a sequence of points such that v_1 is the lowest point, and $v_2, \ldots, v_k$ are in clockwise sorted order around v_1. If for all $1 < i \leq k$, (v_{i-1}, v_i, v_{i+1}) is a right turn, where $v_{k+1} = v_1$, then V are the ordered vertices of a convex hull.*

Proof. By definition V are the ordered vertices of a convex hull if V represents a simple closed convex chain. First, because the vertices in $V = \{v_1, \ldots, v_k\}$ are given in clockwise sorted order around v_1, the closed chain V must be simple (i.e. when rotating a ray from v_1, the edges of the chain always cross it in the same direction). In order for the chain to be a closed convex chain, it must make a right turn at every vertex. We are already explicitly given that a right turn is made at every vertex except for v_1. To see why (v_k, v_1, v_2) is a right turn, observe that v_1 is lower than both v_k and v_2, and moreover v_k comes after v_2 in clockwise order about v_1. These two facts combined imply the angle $\angle v_k v_1 v_2$ is $< \pi$ (i.e. the angle subtended by rotating $v_1 v_2$ clockwise about v_1 to $v_1 v_k$), that is a right turn. □

Given the above discussion about breaking the cost function into cones and slabs according to Equation 3.2, the proof of correctness of Algorithm 1 is now fairly straightforward.

Lemma 16 *Given an instance P, R, f, k of Problem 2 in the plane, Algorithm 1 computes the optimal solution cost, namely $\min_{Q \subseteq R, |Q| \leq k} f(Q, P)$.*

Proof. For any $s \in R$, we now argue $cost_s = \min_{v \in R_s}(sum_{start}(s, v) + \textsc{RecAlg}(s, k-1, s, v))$ is the minimum cost k length convex hull with lowest point s. This will imply Wrapper computes the optimum solution as it takes the minimum of this quantity over all $s \in R$.

Suppose that $cost_s$ is not infinite. By the structure of the recursive algorithm, this can only happen if in each recursive call determining $cost_s$ that *best* is not infinite. The places where *best* can be set to a non-inifinite value are lines 4 and 9, and if the return value of *best* is set by line 4 then this represents a terminal call. Moreover, observe that executing line 4 or 9 requires satisfying a right turn check on the proceeding line. Thus there must have been a sequence of recursive calls made with a corresponding sequence of vertices $V = \{v_1 = s, v_2, \ldots, v_\kappa\}$ such that for all $1 < i \leq \kappa$, $right(v_{i-1}, v_i, v_{i+1}) = true$ (where $v_{\kappa+1} = v_1$), which by Lemma 15 implies V are the ordered vertices of a convex hull. (Note that s being lowest is enforced by considering only R_s, and the clockwise ordering of V is enforced the ordering of the for loops.) Moreover, we have $cost_s = sum_{start}(s, v_2) + \textsc{RecAlg}(s, k-1, s, v_2)$, and from line 9 for all $1 < i < \kappa$ we have $RecAlg(s, k-i+1, v_{i-1}, v_i) = sum_{cone}(v_{i-1}, v_i, v_{i+1}) + sum_{slab}(v_i, v_{i+1}) + RecAlg(s, k-i, v_i, v_{i+1})$, and from line 4 we have $RecAlg(s, k-\kappa+1, v_{\kappa-1}, v_\kappa) = sum_{cone}(v_{\kappa-1}, v_\kappa, s) + sum_{end}(v_{k'}, s)$. Thus putting all these equations together we have

$$cost_s = sum_{start}(s, v_2) + \sum_{i=2}^{\kappa-1}(sum_{cone}(v_{i-1}, v_i, v_{i+1}) + sum_{slab}(v_i, v_{i+1})) + (sum_{cone}(v_{\kappa-1}, v_\kappa, s) + sum_{end}(v_\kappa, s)) = f(V, P)$$

where the last equality follows from Equation 3.2. Thus if $cost_s$ is not infinite then we know it represents the true cost of some valid set of convex hull vertices V. Conversely, by a similar logic it is easy to see that $cost_s$ is never infinite since for the ordered sequence of vertices of any convex hull all the right turn checks will be satisfied and in the algorithm when looking for the next vertex we try all possible vertices that remain in the sorted order. (Note ∞ may be returned if there

Algorithm 1 Recursive Algorithm

```
1:  function RecAlg(s, k', u, v)
2:      best = ∞
3:      if right(u, v, s) then
4:          best = sum_cone(u, v, s) + sum_end(v, s)
5:      if k' = 1 then
6:          return best
7:      for w ∈ R_s after v in clockwise order do
8:          if right(u, v, w) then
9:              best = min{best, sum_cone(u, v, w) +
                sum_slab(v, w) + RecAlg(s, k' − 1, v, w)}
10:     return best

11: function Wrapper(R, P, k)
12:     best = ∞
13:     for s ∈ R do
14:         for v ∈ R_s in clockwise order do
15:             best = min{best, sum_start(s, v) +
                RecAlg(s, k − 1, s, v)}
16:     return best
```

is no non-trivial convex hull, i.e. if s is the highest vertex in R, a case which can be treated separately.) Thus what remains is to argue that the output cost and vertices selected correspond to a minimimal cost solution, however, this is immediate from the above. Specifically, let $\mathcal{V}_i$ be the set of all clockwise ordered convex hull vertices such that all have the same prefix $\{v_1, \ldots, v_i\}$. Then $\min_{V \in V_i} f(V, P)$ is determined by selecting $\{v_{i+1}, \ldots v_k\}$ so as to minimize the cone and slab sums they determine, which as argued above is precisely what lines 9 and 4 do. In particular, because the cones and slabs define an ordered partition of P, minimizing their cost over the remaining vertices, does not affect the cone and slab cost determined by the previously selected vertices, and thus the recursive algorithm correctly returns the minimal cost overall. □

As the correctness of our approach is established by the above lemma, the proof of the following theorem mainly focuses on running time. The proof saves roughly an $O(m)$ factor over the naive time bound by using sweeping both to batch dynamic programming table entries together and to implicitly precompute the sum_{cone} values.

Theorem 5. *Given an instance P, R, f, k of Problem 2 in the plane, it can be solved in $O(m^3k + m^2n + mn\log(n))$ time, where $n = |P|$ and $m = |R|$.*

Proof. First, observe that the recursive Algorithm 1 can easily be turned into a dynamic program, as the k' parameter strictly decreases in each recursive call. Moreover, it is easy to modify the code such that it returns the actual vertices instead of just the cost of the hull.

The correctness of this algorithm follows from Lemma 16. For the running time, first observe that for every vertex $s \in R$ we can compute R_s and the clockwise sorted order of all points in R around s, in $O(m^2 \log m)$ time. So assume this is done initially, and moreover assume for now that all the cone and slab sums have been precomputed. The dynamic program will compute the value of $\textsc{RecAlg}(s, k', u, v)$ for each quadruple (s, k', u, v). Naively this takes $O(m)$ time per quadruple since the for loop on line 7 requires a table lookup for each point in R. Thus overall the dynamic program takes $O(m^4k)$ time as the table size is $O(m^3k)$. However, we can save an $O(m)$ factor in the running time by instead computing for each triple $(s, k', \cdot, v)$, the entire column of u values in $O(m)$ time as follows.

Fix s, k', and v. Define $cost(u, v, w) = sum_{cone}(u, v, w) + sum_{slab}(v, w) + \textsc{RecAlg}(s, k' - 1, v, w)$. For any $u \in R_s$ coming before v in the clockwise order about s, the recursive algorithm computes $\textsc{RecAlg}(s, k', u, v) = \min_{w \in Y(u)} cost(u, v, w)$, where $Y(u)$ is the set of points $w \in R_s$ such that $right(u, v, w) = true$ and moreover w is after v in the clockwise order about s. Specifically, this is all points in the region determined by sweeping the ray from v to s counterclockwise until it hits the line passing through u and v. See Figure A.1. So let $u_1, \ldots, u_z$ be the vertices in R_s coming before v in the clockwise order about s, but labelled by their counterclockwise order about v. Then $Y(u_i) \subseteq Y(u_{i+1})$, and in particular $S_{i+1} = Y(u_{i+1}) \setminus Y(u_i)$ are the set of points in the wedge lying between the line though u_i and v and the line through u_{i+1} and v (again see Figure A.1). Observe that the S_i are disjoint sets, and moreover,

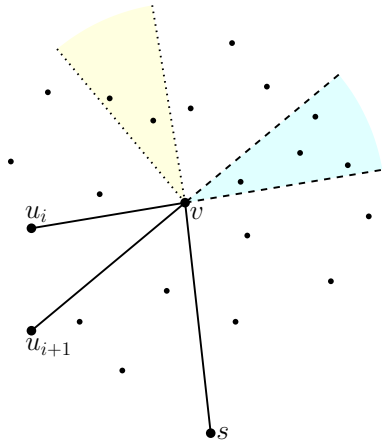


Figure A.1: S_{i+1} with 4 points shown shaded blue on the right. The two points determining $x = sum_{cone}(u_{i+1}, v, \cdot) - sum_{cone}(u_i, v, \cdot)$ shown in shaded yellow on the left.

$$\textsc{RecAlg}(s, k', u_{i+1}, v) = \min_{w \in Y(u_{i+1})} cost(u_{i+1}, v, w)$$
$$= \min\{\min_{w \in S_{i+1}} cost(u_{i+1}, v, w), \min_{w \in Y(u_i)} cost(u_{i+1}, v, w)\}$$

Observe that $\min_{w \in Y(u_i)} cost(u_{i+1}, v, w) = \min_{w \in Y(u_i)} cost(u_i, v, w) + x$ for a fixed value x that does not depends on w. Namely, $x = sum_{cone}(u_{i+1}, v, \cdot) - sum_{cone}(u_i, v, \cdot)$ (see Figure A.1), as the cone sum is the only term in $cost(u, v, w)$ depending on u. Then given we already computed $\textsc{RecAlg}(s, k', u_i, v) = \min_{w \in Y(u_i)} cost(u_i, v, w)$, by the above equation the time to compute $\textsc{RecAlg}(s, k', u_{i+1}, v)$ is proportional to just $|S_i|$. Thus as the S_i are disjoint, this takes $O(m)$ time over all the u_i, resulting in $O(m^3k)$ time for the entire dynamic program.

Now we must consider the time to precompute the cone and slab sums. For any pair $u, v \in R$, $sum_{slab}(u, v)$ can be computed in $O(n)$ time by scanning the points in P to see which fall in the slab, and thus for all pairs in R the sum slab cost can be computed in $O(m^2n)$ time. As $sum_{cone}(u, v, w)$ is determined by three vertices in R, similarly computing these values would take $O(m^3n)$ time, however, we now argue that they can be implic-

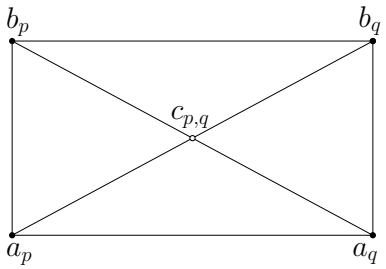


Figure B.1: The edge (a_p, a_q) with added points in the added dimension

itly computed more efficiently as follows. First, fix any vertex $v \in R$. Recal the boundary of $cone(u, v, w)$ is determined by the rays $r_r(u, v)$ and $r_l(v, w)$ (defined above). So let U and W be the sets of all vertices that come before and after v in the clockwise sorted order about s, respectively, and let $R_r = \cup_{u \in U}\ r_r(u, v)$ and $R_l = \cup_{w \in W}\ r_l(v, w)$. Now sort all the vectors in $R_r \cup R_l \cup P$ in clockwise order around v, starting from the first vertex occurring after the negative y-axis direction. Now walk through the vertices in order maintaining a rolling sum, which initially is zero. If the next vertex w is in P then we add $g_v(||v - w||)$ to the sum, otherwise if $w \in R_r$ then we assign the current sum as $value_r(w)$ and if $w \in R_l$ we assign the current sum as $value_l(w)$. Observe that given vertices $u, w \in R$ where u comes before v and w comes after v in clockwise order about s, that $sum_{cone}(u, v, w) = value_l(w) - value_r(v)$. Thus while we do not explicitly compute $sum_{cone}(u, v, w)$ for all triples, by computing all of the $value_l$ and $value_r$ values, then by taking a difference of two such values in constant time we have access to $sum_{cone}(u, v, w)$. This takes $O((n + m) \log(n + m))$ time per vertex in R and thus for all vertices in R takes $O(m(n + m) \log(n + m))$ time. Thus precomputing all the slab sums and implicitly precomputing all the cone sums overall takes $O(m^2 n + m^2 \log(m) + mn \log(n))$ time. Thus total running time of the entire algorithm is $O(m^3 k + m^2 n + mn \log(n))$. □

B Proof from Section 4

Theorem 9. *Problem 7 is NP-hard for $d \geq 4$, $f = \mathsf{sd}$, and $P = R$.*

Proof. We give a polynomial time reduction from Problem 6. Let $T = (V, E)$ and k be an instance of Problem 6. For any edge $e \in E$, let ℓ_e denote its length and m_e its midpoint. Define the quantity $h = \min\{\ell, h_1, h_2\}$, where $\ell = \min_{e \in E} \ell_e$, $h_1 = \min_{p \in V} ||p - \mathcal{CH}(V \setminus \{p\})||$, and $h_2 = \min_{(e_1, e_2) \in E} ||m_{(e_1, e_2)} - \mathcal{CH}(V \setminus \{e_1, e_2\})||$. Then for each point $p = (p_x, p_y, p_z) \in V$, define the points $a_p = (p_x, p_y, p_z, 0)$ and $b_p = (p_x, p_y, p_z, h)$, and for each edge $(p, q) \in E$ define the point $c_{p,q} = (\frac{p_x+q_x}{2}, \frac{p_y+q_y}{2}, \frac{p_z+q_z}{2}, \frac{h}{2})$. See Figure B.1. Define the sets $A = \{a_p \mid p \in V\}$, $B = \{b_p \mid p \in V\}$, and $C = \{c_{p,q} \mid (p, q) \in E\}$. Intuitively, we wish to give all points in B unit weight, and give n^2 weight to all points in A and C. To accomplish this let A' and C' be the multi-sets consisting of n^2 copies of all points in A and C, respectively. Our instance of Problem 7 in $\mathbb{R}^4$ is defined by $P = R = A' \cup B \cup C'$, $k_0 = k + n$, and $\varepsilon = nh$. Note that any solution to Problem 7 containing a point from A' (or C'), does not change in cost if we add one of its duplicates or exchange it for a duplicate. Thus we can assume the optimal solution does not select duplicates, and so below we write $A \subset P$ and refer to selecting points from A.

Let $W \subseteq V$ be a vertex cover of size k for the given instance of Problem 6, and let $B(W) = \{b = (p, h) \in B \mid p \in W\}$. The claim is that $B(W) \cup A$ is a solution to our instance of Problem 7 with cost $\leq \varepsilon$. First, observe that $|B(W) \cup A| = k + n = k_0$ as required. Next, observe that naturally this gives zero error to all points in A' and $B(W)$. The same is true for any point $c_{p,q} \in C'$. To see this observe that since W is a vertex cover, it must contain at least one of p or q. Without loss of generality suppose it contains q, in which case $b_q \in B(W)$. Thus $B(W) \cup A$ contains both b_q and a_p, and since $c_{p,q}$ is defined as the midpoint on the segment between b_q and a_p, it is in their convex hull, i.e. it is covered with zero error. Thus the error can only come from points in $B \setminus B(W)$, however, the error for these points is easily upper bounded by $\varepsilon = nh$, as $|B \setminus B(W)| \leq n$ and since for any point $b_p \in B$ we have $||b_p - a_p|| = h$ and a_p is in our solution.

Now let Q be a solution to Problem 7 with k_0 points and error $\leq \varepsilon$. We first argue that $A \subseteq Q$. Suppose otherwise that some point $a_0 \in A$ is not in Q. We now lower bound $||a_0 - \mathcal{CH}(Q)||$. Specifically, we will assume $Q = P \setminus \{a_0\}$, as this minimizes $||a_0 - \mathcal{CH}(Q)||$ over all possible Q. Observe, that $c_{p,q} \in \mathcal{CH}(Q)$ for any point $c_{p,q} \in C$, since $b_p, b_q \in Q$, and at least one of a_p or a_q is in Q. Thus every point in $\mathcal{CH}(Q)$ either lies in $\mathcal{CH}(A \setminus \{a_0\})$, in $\mathcal{CH}(B)$, or on a segment between a point of $\mathcal{CH}(A \setminus \{a_0\})$ and $\mathcal{CH}(B)$. Let α, β be the closest points to a_0 in $\mathcal{CH}(A \setminus \{a_0\})$ and $\mathcal{CH}(B)$, respectively. Then by the definition of h, $||a_0 - \alpha|| \geq h$, and $||a_0 - \mathcal{CH}(B)|| = h$. Moreover, it is not hard to see that the closest segment, between a point of $\mathcal{CH}(A \setminus \{a_0\})$ and $\mathcal{CH}(B)$, to a_0 is the segment between α and β. Thus the distance form a_0 to $\mathcal{CH}(Q)$ is at least $(\sqrt{h^2 + h^2})/2 = h/\sqrt{2}$. Since A' contains n^2 copies of a_0, the error of Q for Problem 7 is at least $n^2 \frac{h}{\sqrt{2}} > \varepsilon$ (for $n \geq 2$). Thus all points in A must have been selected.

Observe that for any point $c_{p,q} \in C$, that $c_{p,q} \in \mathcal{CH}(a_p, a_q, b_p)$ and $c_{p,q} \in \mathcal{CH}(a_p, a_q, b_q)$, see Figure B.1. That is, since $A \subseteq Q$, if a point $c_{p,q}$ is in Q, exchanging it for either b_p or b_q can only enlarge $\mathcal{CH}(Q)$. Thus with-

out loss of generality we assume Q contains no points from C. Moreover, for any $c_{p,q} \in C$, at least one of b_p or b_q is in Q, since otherwise $c_{p,q} \notin \mathcal{CH}(Q)$, in which case by the same argument as above for a_0, we have $||c_{p,q} - \mathcal{CH}(Q)|| \geq (\sqrt{(h/2)^2 + (h/2)^2})/2 = h/\sqrt{8}$. Since C' contains n^2 copies of $c_{p,q}$, the total error is then at least $n^2 \frac{h}{\sqrt{8}} > \varepsilon$ (for $n \geq 3$), a contradiction. Thus for every point $c_{p,q}$ at least one of b_p or b_q is in Q, or equivalently $W = \{p \mid b_p \in Q\}$ is a vertex cover of E. Moreover, it must be that $|W| = k$, as $k_0 = n + k$ and all n points of A were selected. Thus all that remains is to argue that the error due to $B \setminus W$ is less than ε (as all other points are in $\mathcal{CH}(Q)$). However, since all points in A are in Q, this error is as most $(n-k)h < nh = \varepsilon$. □

C Proof from Section 5

Lemma 12. *Given an instance $P, R \subset \mathbb{R}^d$ and k of Problem 2, where $f = \mathsf{sd}$ and d is a constant, for any subset $Q \subseteq R$ such that $Z(Q) \geq 0$, there exists a d-simplex Δ such that $Z(Q \cup \Delta) \leq \left(1 - \frac{1}{c \cdot k^{\lfloor d/2 \rfloor}}\right) Z(Q)$, where c is a constant.*

Proof. Let $\{\Delta_1, \Delta_2, \ldots, \Delta_\ell\}$ be the d-simplices of the d-dimensional triangulation of $\mathcal{CH}(\mathsf{opt}_{\mathsf{sd}})$ with the minimum number of d-simplices. It is known that $\ell \leq c \cdot k^{\lfloor d/2 \rfloor}$, where c is a constant (using for example the bottom vertex triangulation of [6]). Let $\{P_1, P_2, \ldots, P_\ell\}$ be the partition of P where $p \in P_i$ if and only if $||p - \Delta_i|| = ||p - \mathcal{CH}(\mathsf{opt}_{\mathsf{sd}})||$. (If the projection is on a common point of more than one simplex, assign one arbitrarily.) Now, rewrite $Z(Q)$ as

$$Z(Q) = \sum_{i=1}^{\ell} \sum_{x \in P_i} \left(||x - \mathcal{CH}(Q)|| - ||x - \mathcal{CH}(\mathsf{opt}_{\mathsf{sd}})|| \right).$$

Let $\mathsf{Avg} := \frac{Z(Q)}{\ell}$ denote the average of $Z(Q)$ over the partitions P_i. Hence, there exists a simplex Δ_j with corresponding partition P_j such that

$$\begin{aligned} Z(Q, P_j) &= \sum_{x \in P_j} \left(||x - \mathcal{CH}(Q)|| - ||x - \mathcal{CH}(\mathsf{opt}_{\mathsf{sd}})|| \right) \\ &\geq \mathsf{Avg} \geq \frac{Z(Q)}{c \cdot k^{\lfloor d/2 \rfloor}}, \end{aligned}$$

where note the last inequality is where we used $Z(Q) \geq 0$. Finally, we have $Z(Q \cup \Delta_j, P_j) = \mathsf{sd}(Q \cup \Delta_j, P_j) - \mathsf{sd}(\mathsf{opt}_{\mathsf{sd}}, P_j) = \mathsf{sd}(Q \cup \Delta_j, P_j) - \mathsf{sd}(\Delta_j, P_j) \leq 0$. Thus,

$$\begin{aligned} Z(Q \cup \Delta_j) &= \left(\sum_{i \in [\ell], i \neq j} Z(Q \cup \Delta_j, P_i) \right) + Z(Q \cup \Delta_j, P_j) \\ &\leq \sum_{i \in [\ell], i \neq j} Z(Q, P_i) = \left(\sum_{i \in [\ell]} Z(Q, P_i) \right) - Z(Q, P_j) \\ &\leq \left(1 - \frac{1}{c \cdot k^{\lfloor d/2 \rfloor}}\right) Z(Q) \end{aligned}$$

□

Fair Covering of Points by Balls

Daniel Lokshtanov* Chinmay Sonar* Subhash Suri* Jie Xue*

Abstract

We consider the problem of covering a multi-colored set of points in $\mathbb{R}^d$ using (at most) k disjoint unit-radius balls chosen from a candidate set of unit-radius balls so that each color class is covered *fairly* in proportion to its size. Specifically, we investigate the complexity of covering the maximum number of points in this setting. We show that the problem is NP-hard even in one dimension when the number of colors is large. On the other hand, for a constant number of colors, we present a polynomial time exact algorithm in one dimension, and a PTAS in any fixed dimension $d \geqslant 2$.

1 Introduction

Given a set P of n points in $\mathbb{R}^d$ each of which is colored by one of t colors, the fair covering problem aims to cover the maximum number of points using k unit-radius balls such that the coverage for each color is in *proportion* to its size. More precisely, let $\mathcal{C}$ be a family of k unit radius balls, c_i be the number of the points of color i that are covered by $\mathcal{C}$, and n_i be the total number of points of color i, for $i \in \{1, \ldots, t\}$. Then we say that the covering $\mathcal{C}$ is *fair* if

$$\lfloor \rho_i \cdot c^* \rfloor \leqslant c_i \leqslant \lceil \rho_i \cdot c^* \rceil$$

for all $i \in \{1, \ldots, t\}$, where $c^* = \sum_{i=1}^{t} c_i$ and $\rho_i = n_i/n$ for $i \in \{1, \ldots, t\}$. Among all *fair* coverings, we want the one that maximizes the total coverage c^*. We note that an empty covering trivially satisfies the fairness condition but covers no points.

Achieving strict fair covering can be computationally hard, so we also define the notion of *approximately fair* covering. A covering $\mathcal{C}$ is called ε-fair for some $\varepsilon \in [0, 1]$, if

$$(1-\varepsilon) \cdot \lfloor \rho_i \cdot c^* \rfloor \leqslant c_i \leqslant (1+\varepsilon) \cdot \lceil \rho_i \cdot c^* \rceil$$

for all $i \in \{1, \ldots, t\}$. The goal of the approximately fair covering problem is then to find an ε-fair covering that maximizes the number of covered points.

The topic of algorithmic *fairness* has received significant attention recently [17, 25, 9, 15, 4, 10, 18, 7], especially with the increasing use of machine learning in policy and decision making. Our paper explores the computational implications of fairness as a constraint in geometric optimization by focusing on the specific problem of covering by unit balls, or equivalently, fixed-radius facility location. The different colors in our input represent different demographic groups and *proportionality* is one of the most basic forms of fairness, requiring that each group's share in the solution is proportional to its size. The proportional fairness can be easily extended to *weighted* sharing by assigning nonuniform weights to different points or color classes and measuring fairness on the overall covered weights. The fair covering problem can also be viewed as *fair clustering* under the k-center measure when each cluster is constrained to have unit radius.

Our Results

In this paper, we investigate the aforementioned (approximately) fair covering problem under the *discreteness* and *disjointness* constraints defined below. We require the balls used in a covering to be chosen from a given candidate set of unit-radius balls (discreteness) and to be pairwise disjoint (disjointness). Formally, the input of the problem consists of a set P of n t-colored points in $\mathbb{R}^d$, a candidate set $\mathcal{B}$ of m unit-radius balls in $\mathbb{R}^d$, and a number k that is the budget of balls to be used. Our goal is to find a (approximately) fair covering for P using at most k disjoint balls in $\mathcal{B}$ that covers the maximum number of points. Our main results are the following:

- We show that there exists an exact algorithm solving the fair covering problem in $\mathbb{R}^1$ in $O(mn^t)$ time. Alternatively, the problem can also be solved in $O(nm^k)$ time (Section 2.1).
- We show that the fair covering problem in $\mathbb{R}^1$ is NP-hard if the number of colors is part of the input. We also show that the problem is W[1]-hard parameterized by the number of covering balls k (Section 2.2).
- For a fixed $d \geqslant 2$ and a fixed number of colors, we present a PTAS for the approximately fair covering problem (Section 3).

*University of California, Santa Barbara, USA
Emails: {daniello, csonar, suri, jiexue}@cs.ucsb.edu

Related Work

The problem of covering points by balls or other geometric shapes has a long history in computational geometry, operations research, and theoretical computer science, due to its natural connections to clustering and facility location problems [3, 14, 20, 23, 24]. It is known that covering a set of two-dimensional points with a minimum number of unit disks is NP-hard, and so is the problem of maximizing the number of points covered by k unit disks [13, 19, 8, 11]. Recently, a number of researchers have considered clustering and covering problems with an additional constraint of fairness. In this setting, the input consists of points belonging to different colors (classes), and the goal is to find a solution where *each cluster* has approximately equal representation of all colors [21, 10, 6, 1, 22]. These formulations are different from our model because we allow individual clusters to be unbalanced as long as *in aggregate* each color receives its fair share. This non-local form of fair representation seems much harder than requiring each cluster to locally meet the balance condition. In another line of work, [5, 15, 2] consider a *colorful* variant of the k-center problem where the goal is to satisfy a *minimum* coverage for each color type. The colorful covering however does not achieve fairness because some color classes can have arbitrarily high representation in the output, as long as other colors meet the minimum threshold. In fact, enforcing the fairness by controlling *both* the lower and the upper bounds of representation seems to be a much harder problem, as suggested by some of our hardness results in one dimension.

2 Fair Covering in One Dimension

We begin by considering the problem in one dimension. Let $P = \{p_1, \ldots, p_n\}$ be a set of n points on the real line each of which belongs to one of the t color classes, and let $\mathcal{B} = \{B_1, \ldots, B_m\}$ be the candidate set of unit intervals on the line. (Technically speaking, a unit-radius ball in one dimension would be an interval of length 2, but a *unit-length* interval seems more natural, so that we shall use unit intervals in the following discussion. Note that the problem with intervals of length 2 is equivalent to the problem with unit intervals by simply scaling the points and the intervals.) Our goal is to cover the maximum number of points using at most k disjoint intervals in $\mathcal{B}$ under the fair covering constraint. We show that an optimal covering can be computed in polynomial time when the number t of colors is fixed, but the problem becomes intractable when t is part of the input.

2.1 A Dynamic Programming Algorithm

For simplicity, we describe our algorithm for $t = 2$ and use red/blue as the two colors for easier reference. The extension to an arbitrary number of colors is straightforward.

Given integers r and b, we define an (r, b)-*covering* to be a subset of $\mathcal{B}$ consisting of disjoint intervals that covers exactly r red and b blue points. An optimal (r, b)-covering is an (r, b)-covering that uses the minimum number of intervals. We solve the fair covering problem by computing an optimal (r, b)-covering for all $r, b \in \{1, \ldots, n\}$. Without loss of generality, we assume that the unit intervals $B_1, \ldots, B_m$ are sorted in the left-to-right order. Let $r(B_i)$ and $b(B_i)$ be the number of the red and blue points covered by B_i, respectively. For each $i \in \{1, \ldots, m\}$, let $\pi_i < i$ be the largest integer such that $B_{\pi_i} \cap B_i = \emptyset$; we assume $\pi_1 = 0$. We make a left-to-right pass over the set of input points and the intervals on the real line, and compute π_i, $r(B_i)$, $b(B_i)$ for all $i \in \{1, \ldots, m\}$.

Define $F[i, r, b]$ as the size of an optimal (r, b)-covering using only intervals in $\{B_1, \ldots, B_i\}$. For the pairs (r, b) such that no (r, b)-covering exists, we set $F[i, r, b] = \infty$. It is easy to see that F satisfies the following recurrence.

Claim 1

$$F[i, r, b] \;=\; \min \left\{ \begin{array}{l} F[i-1, r, b] \\ 1 + F[\pi_i, r - r(B_i), b - b(B_i)] \end{array} \right\}$$

The above recurrence immediately allows us to compute the table F using dynamic programming, which is shown in Algorithm 1. The base case for the dynamic program is $F[i, 0, 0] = 0$ for all $i \in \{1, \ldots, m\}$ and $F[0, r, b] = \infty$ for all $r, b \in \{1, \ldots, n\}$.

Algorithm 1: Computing the F-table

Input: $P, \mathcal{B}$

```
Compute π_i, r(B_i), b(B_i) for i ∈ {1, ..., m}
Initialize m × r × b sized table with value ∞
for i ∈ {0, ..., m}; r, b ∈ {0, ..., n} do
    F[i, r, b] ←
      min{F[i−1, r, b], 1 + F[π_i, r − r(B_i), b − b(B_i)]}
end
return F
```

Lemma 2 *Algorithm 1 can be implemented in worst-case time $O((n + m) \log(n + m) + mn^2)$.*

Proof. Sorting P and $\mathcal{B}$ takes $O((n + m) \log(n + m))$ time. Computing $\pi_i, r(B_i), b(B_i)$ for all $i \in \{1, \ldots, m\}$ takes additional linear time. After that the F-table can be computed in $O(mn^2)$ time. □

Once the F-table is computed, we can solve the fair covering problem by checking all entries in the table for which the (r, b)-covering is fair and has $F[m, r, b] \leqslant k$. Among all such valid pairs, we return the pair (r^*, b^*) with the maximum $r^* + b^*$. Clearly, $c^* = r^* + b^*$ is the optimum of the problem instance. We therefore have the following result.

Theorem 3 *The fair covering problem in $\mathbb{R}^1$ with $t = 2$ colors can be solved in $O((n+m)\log(n+m)+mn^2)$ time.*

The dynamic program easily extends to the case of $t > 2$ colors, by using a $(t + 1)$-dimensional DP table.

Theorem 4 *The fair covering problem in $\mathbb{R}^1$ can be solved in $O((n + m)\log(n + m) + mn^t)$ time.*

Remarks. Recall that the fair covering problem we investigate is defined with the discreteness and disjointness constraints. In fact, the problem without each of these two constraints can also be solved using similar dynamic programming approaches. We omit the details here because our main focus is the problem with the discreteness and disjointness constraints.

2.2 NP and W[1]-Hardness of the Fair Covering

In this section, we show that the one-dimensional fair covering problem is NP-hard if the number of colors t is large. We also show that the problem is W[1]-hard parameterized by the number of intervals k.

Theorem 5 *The one-dimensional fair covering problem with $\Omega(n)$ colors is NP-hard.*

Proof. We reduce the well-known Exact Cover problem [16] to our problem. Given a ground set $\mathcal{U}$, a family $\mathcal{F}$ of subsets of $\mathcal{U}$, and an integer ℓ, the Exact Cover problem is to decide if there exists a $\mathcal{S} \subseteq \mathcal{F}$ of size ℓ that contains each element of $\mathcal{U}$ exactly once. The construction is described below.

Construction. Given an instance of Exact Cover with $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$, $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$, and an integer ℓ, we construct a set of points P, and a set of centers $\mathcal{M}$ as follows. The i^{th} element of $\mathcal{U}$ is associated with color i; thus, there are n color classes. We also introduce an additional color 0, which we call *special*. The set of points is organized in the following three groups.

1. *Basic Points:* For each set $S_i \in \mathcal{F}$, we introduce $|S_i|$ points, placed arbitrarily within the interval $[3i, 3i+1)$. Each point has the color of its element. The intervals corresponding to S_i and S_j, $i \neq j$, are distance 2 apart, which ensures that any unit interval of $\mathcal{B}$ can cover points of at most one such group.
2. *Balancers:* We add extra points for each color i to ensure that all colors $i = 1, 2, \ldots, n$ end up with the same number of points. Specifically, let f^* be the maximum number of sets to which an element belongs, and let f_i be the number of sets containing the element u_i. We introduce $f^* - f_i$ points of color i in the interval $[3(m + i), 3(m + i) + 1)$.
3. *Enforcers:* Finally, we introduce ℓf^* points of color 0 (special color), at locations $3(m + n + 1), 3(m + n+2), \ldots, 3(m+n+\ell f^*)$. These are needed in our construction to enforce the fair covering condition. Refer figure 1.

Figure 1: Constructed fair covering instance for an Exact Cover instance $\mathcal{U} = \{1, 2, 3\}$, $\mathcal{F} = \{(1, 3), (2), (1, 2)\}$, $\ell = 2$. We introduce red (1), green (2), and blue (3) colors corresponding to the elements in the universe, and we also introduce cyan as the special color. First five points are introduced in the basic points group. Since $f^* = 2$ (where f^* is a maximum number of sets to which an element of $\mathcal{U}$ belongs to), next, we introduce one blue point so that each color except for cyan has exactly two points. At last, we introduce 4 cyan points as enforcers (since $f^* = \ell = 2$).

Finally, the set of centers $\mathcal{M}$ is defined as follows.

- For each $S_i \in \mathcal{F}$, we add a center at $3i+1/2$, which allows all points of that group to be covered by one unit interval.
- Each enforcer point is also a center. We do not need centers for the balancers—their role is primarily to make all color classes have equal size.

Finally, we fix the number of covering intervals to be $k = 2\ell$.

We now argue that the Exact Covering instance is a yes instance if and only if our fair covering instance admits a k-covering with at least $n + \ell$ points.

For the forward direction of the proof, suppose $\mathcal{S} \subseteq \mathcal{F}$ is an exact cover of size ℓ, and $\mathcal{T} = \{i \mid S_i \in \mathcal{S}\}$ be the set of indices. Then we build a covering $\mathcal{C}$ as follows. We place first ℓ intervals centered at $3i + 1/2$ for $i \in \mathcal{T}$, and the remaining ℓ intervals are placed at $3(m + n + j)$ for $j = 1, 2, \ldots, \ell$ covering one special colored point each. Since $\mathcal{S}$ is an exact cover, $\mathcal{C}$ contains exactly $n + \ell$ points. The covering is also fair, since

all the colors $i = 1, 2, \ldots, n$ have the same number of points f^*, and the special color 0 has ℓf^* points. In the covering, each of the color classes $i = 1, 2, \ldots, n$ has one covered point and the special color has ℓ points.

For the reverse direction, let $\mathcal{C}$ be the fair covering with at least $n + \ell$ points. We observe that a fair covering necessarily contains the same number of points, say p, for each color $i = 1, 2, \ldots, n$, and contains exactly ℓp points of the special color. For $p = 2$, to cover 2ℓ special colored points only, we need all 2ℓ intervals. Hence, for any *fair* covering, we get $p < 2$. This implies that for the covering $\mathcal{C}$, $p = 1$ to meet the overall covering requirement. Since, we need ℓ intervals to cover ℓ special colored points, it is easy to see that the remaining ℓ intervals cover exactly one point of every other color. Hence, the intervals covered corresponds to an Exact Cover. □

In the reduced instance above, the number of intervals is dependent only upon the size of the Exact Cover (ℓ). The Exact Cover problem is known to be W[1]-hard parameterized by ℓ [12]. Hence, the analogous results for the fair covering problem is summarized as follows:

Theorem 6 *The fair covering problem is* W[1]*-hard parameterized by the number of covering balls* (k).

In dimensions $d \geqslant 2$, the maximum coverage problem is NP-hard [13], and W[1]-hard [19], *even without the fairness constraint.*

3 A PTAS for Fair Covering in d Dimensions

In this section, we describe a PTAS for the approximately fair covering problem in any fixed dimension d. Specifically, given an approximate factor $\varepsilon \in [0, 1]$, we want to compute an ε-fair covering of P (using at most k disjoint balls in $\mathcal{B}$) such that the number of the points covered is at least $(1-\varepsilon)\cdot\mathsf{opt}$, where opt is the size of an optimal fair covering of P. In other words, the approximation is bi-criteria: one criterion is on the fairness of the covering while the other one is on the quality of the solution (i.e., the number of the points covered). For the simplicity of exposition, we describe the algorithm in two dimensions ($d = 2$) and for two colors ($t = 2$). The extension to higher dimensions and the general case of $t > 2$ colors is straightforward.

3.1 Shifted Partitions & Approximate Covering

When solving the fair covering problem in $\mathbb{R}^1$, we were able to compute an optimal (r, b)-covering for any (r, b) pair. This seems quite difficult in higher dimensions, and so we resort to solving an approximate version of this problem as follows. We want to compute a table $\Gamma[1 \ldots n, 1 \ldots n]$ of integers such that for each pair (r, b), we have the following:

1. $\Gamma[r, b]$ is at least the size of an optimal (r, b)-covering, and
2. there exists $r^* \in [(1 - \varepsilon)r, r]$ and $b^* \in [(1 - \varepsilon)b, b]$ such that $\Gamma[r^*, b^*]$ is at most the size of an optimal (r, b)-covering.

For convenience, we call such a table Γ an *ε-approximate covering table* (ε-ACT) for the instance $(P, \mathcal{B})$. Note that to solve the approximately fair covering problem, it suffices to compute an ε-ACT.

Lemma 7 *Given an ε-ACT Γ for $(P, \mathcal{B})$, one can solve the approximately fair covering problem in polynomial time.*

Proof. Suppose an optimal fair covering covers r_0 red points and b_0 blue points. We call a pair (r, b) with $r, b \in \{1, \ldots, n\}$ *feasible* if **(1)** an (r, b)-covering is fair and **(2)** there exists $r^* \in [(1-\varepsilon)r, r]$ and $b^* \in [(1-\varepsilon)b, b]$ such that $\Gamma[r^*, b^*] \leqslant k$. We compute all feasible pairs, which can clearly be done in polynomial time given Γ, and find the feasible pair (r, b) that maximizes $r + b$. By definition, we can find $r^* \in [(1 - \varepsilon)r, r]$ and $b^* \in [(1-\varepsilon)b, b]$ such that $\Gamma[r^*, b^*] \leqslant k$. Note that an (r^*, b^*)-covering is ε-fair. Furthermore, $r+b \geqslant \mathsf{opt}$ since (r_0, b_0) is feasible, hence $r^* + b^* \geqslant (1 - \varepsilon) \cdot \mathsf{opt}$. Because Γ is an ε-ACT, there exists an (r^*, b^*)-covering using at most k (disjoint) disks in $\mathcal{B}$. Therefore, $r^* + b^*$ is a $(1 - \varepsilon)$-approximate solution for the approximately fair covering problem. □

In order to compute an ε-ACT Γ, we use the shifting technique [14]. Let $h = h(\varepsilon)$ be an integer parameter to be determined later. For an integer $i \in \mathbb{Z}$, let $\square_{i,j}$ denote the $h \times h$ square $[i, i+h] \times [j, j+h]$; we say $\square_{i,j}$ is *nonempty* if it contains at least one point in P. We first compute the index set $I = \{(i, j) : \square_{i,j} \text{ is nonempty}\}$. This can be easily done in time polynomial in n and h, by computing for each $p \in P$, the $O(h^2)$ squares $\square_{i,j}$ that contains p. For each $(i, j) \in I$, define $P_{i,j} = P \cap \square_{i,j}$ and $\mathcal{B}_{i,j} = \{B \in \mathcal{B} : B \subseteq \square_{i,j}\}$. In the next step, we compute a 0-ACT $\Gamma_{i,j}$ for each $(P_{i,j}, \mathcal{B}_{i,j})$ with $(i, j) \in I$. We will show later in Section 3.2 how to compute $\Gamma_{i,j}$ in $(n_{i,j} + m_{i,j})^{O(h^2)}$ time, where $n_{i,j} = |P_{i,j}|$ and $m_{i,j} = |\mathcal{B}_{i,j}|$. At this point, let us assume we have the 0-ACTs $\Gamma_{i,j}$ and finish the description of our PTAS. We have the following key observation.

Lemma 8 *Let $\{P_1, \ldots, P_s\}$ be a partition of P and $\mathcal{B}_1, \ldots, \mathcal{B}_s \subseteq \mathcal{B}$ be disjoint subsets such that the disks*

in $\mathcal{B}_i$ do not cover any points in $P \backslash P_i$. Given 0-ACTs for $(P_1, \mathcal{B}_1), \ldots, (P_s, \mathcal{B}_s)$, we can compute a 0-ACT for $(P, \bigcup_{i=1}^{s} \mathcal{B}_i)$ in polynomial time.

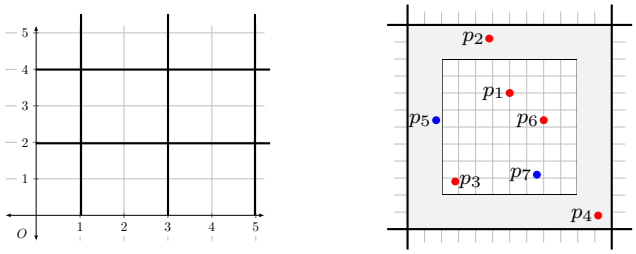
Proof. Computing a 0-ACT for $(P, \bigcup_{i=1}^{s} \mathcal{B}_i)$ is equivalent to computing for all pairs (r, b) the size of the smallest (r, b)-covering of $(P, \bigcup_{i=1}^{s} \mathcal{B}_i)$. Since the disks in $\mathcal{B}_i$ can only cover the points in P_i, the entire problem instance can be divided into independent sub-problems $(P_1, \mathcal{B}_1), \ldots, (P_s, \mathcal{B}_s)$. This allows us to solve the problem in polynomial time using dynamic programming; see Algorithm 2. □

Algorithm 2: Computing the 0-ACT

Input: $\Gamma_1, \ldots, \Gamma_s$, where Γ_i is a 0-ACT for $(P_i, \mathcal{B}_i)$

Initialize a $s \times n \times n$ table F with value ∞
for $t \in \{1, \ldots, s\}$; $r, b \in \{1, \ldots, n\}$ **do**
  $F[t, r, b] \leftarrow \min_{\substack{0 \leqslant r' \leqslant r \\ 0 \leqslant b' \leqslant b}} \{\Gamma_t[r', b'] + F[t-1, r-r', b-b']\}$
end
$\Gamma^*[r, b] = F[s, r, b]$ for all $r, b \in \{1, \ldots, n\}$.
return Γ^*

For $x, y \in \{0, \ldots, h-1\}$, let $L_{x,y}$ be the set of all integer pairs (i, j) such that $i \bmod h = x$ and $j \bmod h = y$ (See Fig. 2a). We write $I_{x,y} = I \cap L_{x,y}$.



(a) (b)

Figure 2: (a) The squares $\square_{i,j}$ for $(i, j) \in L_{1,0}$, with $h = 2$. (b) An illustration of the boundary points. The outer square is $\square_{i,j}$ and the inner square is $[i+2, i+h-2] \times [j+2, j+h-2]$, with $h = 12$. The points in the gray region (i.e., p_2, p_4, p_5) are the boundary points in $\square_{i,j}$.

Lemma 9 *For all $x, y \in \{0, \ldots, h-1\}$, the squares $\square_{i,j}$ for $(i, j) \in I_{x,y}$ are interior-disjoint and cover all points in P.*

Proof. Note that the squares $\square_{i,j}$ for $(i, j) \in L_{x,y}$ are interior-disjoint and cover the entire plane $\mathbb{R}^2$ (see Figure 2a for an example). It directly follows that the squares $\square_{i,j}$ for $(i, j) \in I_{x,y}$ are interior-disjoint. Consider a point $p \in P$ and let $(i, j) \in L_{x,y}$ such that $p \in \square_{i,j}$. Clearly, $(i, j) \in I$ as $\square_{i,j}$ is nonempty and hence $(i, j) \in I_{x,y}$. Therefore, all points in P are covered by the squares $\square_{i,j}$ for $(i, j) \in I_{x,y}$. □

Fix $x, y \in \{0, \ldots, h-1\}$. We know by Lemma 9 that $\{P_{i,j} : (i, j) \in I_{x,y}\}$ is a partition of P and the collections $\mathcal{B}_{i,j}$ for $(i, j) \in I_{x,y}$ are disjoint. Furthermore, the disks in $\mathcal{B}_{i,j}$ do not cover any point in $P \backslash P_{i,j}$. Therefore, we can apply Lemma 8 to compute a 0-ACT $\Gamma^{(x,y)}$ for $(P, \bigcup_{(i,j) \in I_{x,y}} \mathcal{B}_{i,j})$ in polynomial time. We do this for all $x, y \in \{0, \ldots, h-1\}$. Finally, we construct the table Γ by setting $\Gamma[r, b] = \min_{x,y \in \{0,\ldots,h-1\}} \Gamma^{(x,y)}[r, b]$. We shall show that Γ is a $\frac{12h-12}{h^2}$-ACT for $(P, \mathcal{B})$. To this end, we introduce some notions. For a point $p \in P$ and a square $\square_{i,j}$, we say p is a *boundary point* in $\square_{i,j}$ if $p \in \square_{i,j}$ and $p \notin [i+2, i+h-2] \times [j+2, j+h-2]$ (See Figure 2b). Now consider some $x, y \in \{0, \ldots, h-1\}$. We say $p \in P$ *conflicts* with the pair (x, y) if p is a boundary point in $\square_{i,j}$ where $(i, j) \in I_{x,y}$ is the (unique) pair such that $p \in \square_{i,j}$. One can easily see that each point $p \in P$ conflicts with exactly $h^2 - (h-2)^2$ pairs (x, y).

Lemma 10 *For any $P' \subseteq P$, there exists some $x, y \in \{0, \ldots, h-1\}$ such that the number of red (resp., blue) points in P' conflicting with (x, y) is at most $\frac{12h-12}{h^2} \cdot n'_{red}$ (resp., $\frac{12h-12}{h^2} \cdot n'_{blue}$), where n'_{red} (resp., n'_{blue}) is the total number of red (blue) points in P'.*

Proof. Define $\delta^{\text{red}}_{x,y}$ (resp., $\delta^{\text{blue}}_{x,y}$) as the number of the red (resp., blue) points in P' that conflict with (x, y). Because any point $p \in P$ conflicts with exactly $h^2 - (h-2)^2$ pairs (x, y), we have

$$\sum_{x=0}^{h-1} \sum_{y=0}^{h-1} \delta^{\text{red}}_{x,y} = n'_{\text{red}}(h^2 - (h-2)^2) = n'_{\text{red}}(4h-4).$$

Therefore, the number of the pairs (x, y) such that $\delta^{\text{red}}_{x,y} \geqslant 3n'_{\text{red}}(4h-4)/h^2$ is at most $h^2/3$. Equivalently, the number of the pairs (x, y) such that $\delta^{\text{red}}_{x,y} < 3n'_{\text{red}}(4h-4)/h^2$ is at least $2h^2/3$. For the same reason, the number of the pairs (x, y) such that $\delta^{\text{blue}}_{x,y} < 3n'_{\text{blue}}(4h-4)/h^2$ is at least $2h^2/3$. Since $2h^2/3 + 2h^2/3 > h^2$, there exists at least one pair (x, y) that simultaneously satisfies $\delta^{\text{red}}_{x,y} < 3n'_{\text{red}}(4h-4)/h^2$ and $\delta^{\text{blue}}_{x,y} < 3n'_{\text{blue}}(4h-4)/h^2$. This completes the proof of the lemma. □

Now we are ready to prove that Γ is a $\frac{12h-12}{h^2}$-ACT.

Lemma 11 *Γ is a $\frac{12h-12}{h^2}$-ACT for $(P, \mathcal{B})$.*

Proof. Set $\eta = \frac{12h-12}{h^2}$. By the definition of a η-ACT, we have to verify that **(1)** $\Gamma[r,b]$ is at least the size of a smallest (r,b)-covering of $(P,\mathcal{B})$ and **(2)** there exist $r^* \in [(1-\eta)r, r]$ and $b^* \in [(1-\eta)b, b]$ such that $\Gamma[r^*,b^*]$ is at most the size of a smallest (r,b)-covering of $(P,\mathcal{B})$. Condition **(1)** is clearly true. Indeed, for all $x,y \in \{0,\ldots,h-1\}$, $\Gamma^{(x,y)}[r,b]$ is the size of the smallest (r,b)-covering of $(P, \bigcup_{(i,j)\in I_{x,y}} \mathcal{B}_{i,j})$ and hence is at least the size of a smallest (r,b)-covering of $(P,\mathcal{B})$. Next, we verify condition **(2)**. Let $\mathcal{B}' \subseteq \mathcal{B}$ be a smallest (r,b)-covering of $(P,\mathcal{B})$ and $P' \subseteq P$ be the points covered by the disks in $\mathcal{B}'$ (hence P' consists of r red points and b blue points). By Lemma 10, there exist $x,y \in \{0,\ldots,h-1\}$ such that the number of red (resp., blue) points in P' conflicting with (x,y) is at most ηr (resp., ηb). Let $\mathcal{B}'' = \mathcal{B}' \cap (\bigcup_{(i,j)\in I_{x,y}} \mathcal{B}_{i,j})$ and $P'' \subseteq P'$ be the points covered by the disks in $\mathcal{B}''$. Suppose P'' consists of r^* red points and b^* blue points. Note that any disk in $\mathcal{B}'\backslash\mathcal{B}''$ can only cover the points in P that conflict with (x,y). Therefore, any point in P' that does not conflict with (x,y) must be contained in P'', which implies that $r^* \in [(1-\eta)r, r]$ and $b^* \in [(1-\eta)b, b]$. Since $\Gamma^{(x,y)}$ is a 0-ACT for $(P, \bigcup_{(i,j)\in I_{x,y}} \mathcal{B}_{i,j})$, we have $\Gamma^{(x,y)}[r^*,b^*] \leqslant |\mathcal{B}''| \leqslant |\mathcal{B}'|$. It follows that condition **(2)** is also true. □

We set h to be the smallest integer such that $\frac{12h-12}{h^2} \leqslant \varepsilon$; clearly, $h = O(1/\varepsilon)$. Then by the above lemma, Γ is an ε-ACT for $(P,\mathcal{B})$. In this way, we obtain a PTAS for the fair covering problem in $\mathbb{R}^2$.

Theorem 12 *There exists a $(1-\varepsilon)$-approximation algorithm for the fair covering problem in $\mathbb{R}^2$ which runs in $n^{O(1)}m^{O(1/\varepsilon^2)}$ time.*

Proof. In our algorithm, the most time-consuming work is the computation of each $\Gamma_{i,j}$ for $(i,j) \in I$, which takes $n_{i,j}^{O(1)} m_{i,j}^{O(h^2)}$ time as claimed before. All the other work can be done in time polynomial in h, n, m. Since $I = O(h^2 n)$, the overall time complexity of our algorithm is $(n+m)^{O(h^2)}$, i.e., $n^{O(1)}m^{O(1/\varepsilon^2)}$. □

The algorithm can be straightforwardly generalized to higher dimensions and the case $t > 2$, resulting in the following theorem.

Theorem 13 *There exists a $(1-\varepsilon)$-approximation algorithm for the t-color fair covering problem in $\mathbb{R}^d$ which runs in $n^{O(t)}m^{O(1/\varepsilon^d)}$ time.*

3.2 Computing the 0-ACTs $\Gamma_{i,j}$

We now discuss the only missing piece in our algorithm above: the computation of the tables $\Gamma_{i,j}$. Recall that $\Gamma_{i,j}$ is a 0-ACT for $(P_{i,j}, \mathcal{B}_{i,j})$. We show that each $\Gamma_{i,j}$ can be computed in $n_{i,j}^{O(1)} m_{i,j}^{O(h^2)}$ time where $n_{i,j} = |P_{i,j}|$ and $m_{i,j} = |\mathcal{B}_{i,j}|$. The key observation is the following.

Lemma 14 *For $r,b \in \{1,\ldots,n_{i,j}\}$, an (r,b)-covering of $(P_{i,j},\mathcal{B}_{i,j})$ is of size at most $\lfloor h^2/\pi \rfloor$.*

Proof. Recall that an (r,b)-covering of $(P_{i,j},\mathcal{B}_{i,j})$ consists of *disjoint* disks in $\mathcal{B}_{i,j}$. All disks in $\mathcal{B}_{i,j}$ are contained in the $h \times h$ square $\square_{i,j}$. The area of $\square_{i,j}$ is h^2 and the area of a unit-disk is π. Therefore, any subset of disjoint disks in $\square_{i,j}$ is of size at most $\lfloor h^2/\pi \rfloor$. □

With the above observation, we can compute $\Gamma_{i,j}$ as follows. We enumerate all subsets of $\mathcal{B}_{i,j}$ of size at most $\lfloor h^2/\pi \rfloor$, and keep the ones that consist of disjoint disks. In this way, we obtain all (r,b)-coverings of $(P_{i,j},\mathcal{B}_{i,j})$ for all $r,b \in \{1,\ldots,n_{i,j}\}$. By checking these coverings one by one, we can find the smallest (r,b)-covering for all $r,b \in \{1,\ldots,n_{i,j}\}$, and hence compute $\Gamma_{i,j}$. The total time cost is $n_{i,j}^{O(1)} m_{i,j}^{O(h^2)}$.

4 Conclusion

In this paper, we introduced a new fair-covering problem, which is motivated by fair representation of multiple demographics in a geometric facility location setting. We proved that the problem is NP-hard even in one dimension when the number of color groups is large. When the number of colors is fixed, we presented a polynomial time exact algorithm in one dimension, and a PTAS in any fixed dimension. Many open problems remain, including whether one can achieve a constant factor approximation significantly faster than our PTAS, and whether the PTAS can be achieved for covering by non-disjoint balls.

References

[1] S. Ahmadian, A. Epasto, R. Kumar, and M. Mahdian. Clustering without over-representation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 267–275, 2019.

[2] G. Anegg, H. Angelidakis, A. Kurpisz, and R. Zenklusen. A Technique for Obtaining True Approximations for k-Center with Covering Constraints. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 52–65. Springer, 2020.

[3] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, 1994.

[4] A. Backurs, P. Indyk, K. Onak, B. Schieber, A. Vakilian, and T. Wagner. Scalable fair clustering. *arXiv preprint arXiv:1902.03519*, 2019.

[5] S. Bandyapadhyay, T. Inamdar, S. Pai, and K. Varadarajan. A Constant Approximation for Colorful k-Center. In *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[6] S. Bera, D. Chakrabarty, N. Flores, and M. Negahbani. Fair algorithms for clustering. In *Advances in Neural Information Processing Systems*, pages 4955–4966, 2019.

[7] L. E. Celis, V. Keswani, D. Straszak, A. Deshpande, T. Kathuria, and N. K. Vishnoi. Fair and diverse DPP-based data summarization. *arXiv preprint arXiv:1802.04023*, 2018.

[8] T. M. Chan and E. Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Computational Geometry*, 47(2):112–124, 2014.

[9] X. Chen, B. Fain, L. Lyu, and K. Munagala. Proportionally Fair Clustering. In *International Conference on Machine Learning*, pages 1032–1041, 2019.

[10] F. Chierichetti, R. Kumar, S. Lattanzi, and S. Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017.

[11] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.

[12] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995.

[13] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information processing letters*, pages 133–137, 1981.

[14] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM (JACM)*, pages 130–136, 1985.

[15] X. Jia, K. Sheth, and O. Svensson. Fair Colorful k-Center Clustering. In *Integer Programming and Combinatorial Optimization*, pages 209–222. Springer, 2020.

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

[17] J. Kleinberg, H. Lakkaraju, J. Leskovec, J. Ludwig, and S. Mullainathan. Human decisions and machine predictions. *The quarterly journal of economics*, 133(1):237–293, 2018.

[18] M. Kleindessner, P. Awasthi, and J. Morgenstern. Fair k-Center Clustering for Data Summarization. In *International Conference on Machine Learning*, pages 3448–3457, 2019.

[19] D. Marx. Efficient approximation schemes for geometric problems? In *European Symposium on Algorithms*, pages 448–459, 2005.

[20] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.

[21] C. Rösner and M. Schmidt. Privacy Preserving Clustering with Constraints. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[22] M. Schmidt, C. Schwiegelshohn, and C. Sohler. Fair coresets and streaming algorithms for fair k-means. In *International Workshop on Approximation and Online Algorithms*, pages 232–251. Springer, 2019.

[23] C. Toregas, R. Swain, C. ReVelle, and L. Bergman. The location of emergency service facilities. *Operations research*, 19(6):1363–1373, 1971.

[24] D. Yang, S. Misra, X. Fang, G. Xue, and J. Zhang. Two-tiered constrained relay node placement in wireless sensor networks: Computational complexity and efficient approximations. *IEEE Transactions on Mobile Computing*, 11(8):1399–1411, 2011.

[25] M. B. Zafar, I. Valera, M. Rodriguez, K. Gummadi, and A. Weller. From parity to preference-based notions of fairness in classification. In *Advances in Neural Information Processing Systems*, pages 229–239, 2017.

Covering Points with Pairs of Concentric Disks*

Anil Maheshwari† Saeed Mehrabi† Sasanka Roy‡ Michiel Smid†

Abstract

In this paper, we study the following problem motivated by applications in wireless local area networks. We are given a set of m pairs of concentric disks in d-dimensional space, $d \in \{1, 2\}$, where each pair consists of one disk with radius one and the other with radius two. We are also given a set of n points such that the union of the m pairs of disks covers all the n points. The goal is to select *exactly* one disk from each pair such that every point is covered by at least one disk and the number of points covered by at least one disk with radius one is maximized; we refer to this as the sDiskCover problem.

When $d = 1$ (i.e., we have m pairs of intervals on the real line), we give an exact algorithm that solves the sDiskCover problem in $O(m^2n)$ time. We also consider a special case of the problem for $d = 1$, and show that it can be solved in $O(mn)$ time. For $d = 2$, we prove that the sDiskCover problem is NP-hard.

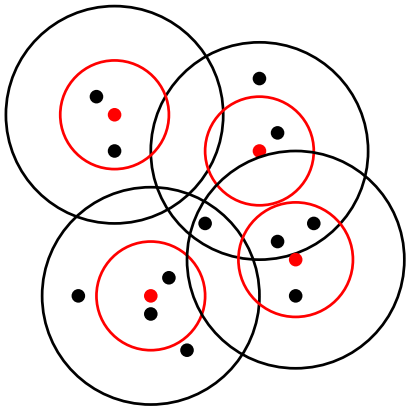


Figure 1: The points in red (resp., black) are the locations of access points (resp., users). The areas within red and black disks centered at each access point denote higher-to-lower frequency disks of the access point. A user can be served with the maximum speed if they are within a red disk and the corresponding access point is activated by that frequency.

1 Introduction

In this paper, we study a problem that is motivated by applications in wireless local area networks (WLANs) [1]. In a WLAN, all the *users* (also called *stations*) receive data from *access points*. An access point can operate exactly one frequency, which can be chosen from many different frequencies at the beginning. When an access point is activated by a single frequency, it covers a circular area inside a disk. Higher frequency has higher speed, but lower coverage in disk area (i.e., covers a disk with smaller radius); see Figure 1 for an example. One can view different frequencies at an access point as concentric disks that are centered at the access point with different radii. Disks with smaller radius have higher frequency, which means the corresponding access point can provide data with higher speed. If a user is within a higher-frequency region of an access point, then they can be supplied data with higher speed; this will correspond to the profit of the service provider who installs the frequency at the access point. The service provider has to provide services to all the users, which might force the service provider to allocate lower frequency at an access point to get a higher coverage area.[1] The objective of the service provider is to increase sum of the total speed provided to the users, which in turn will maximize the profit made by the service provider. In this paper, we formalize this problem with two types of frequencies.

Problem statement. Let $d \in \{1, 2\}$. Then, an *object* in d-dimensional space is a pair of disks, a disk with radius one and a disk with radius two, such that the disk with radius one is entirely contained in the one with radius two. For an object i, we call the disk of i with radius one (resp., two) the *small disk* (resp., *big disk*) of i and denote it by $\mathsf{sDisk}(i)$ (resp., $\mathsf{bDisk}(i)$).

Consider a set of $n > 0$ points $p_1, \ldots, p_n$ and a set of $m > 0$ objects in d-dimensional space for some $d \in \{1, 2\}$. Then, the objective of the sDiskCover problem is to select *exactly* one disk from each object such that

*This work is supported in part by NSERC.

†School of Computer Science, Carleton University, Ottawa, Canada. anil@scs.carleton.ca, saeed.mehrabi@carleton.ca, michiel@scs.carleton.ca.

‡ACMU, Indian Statistical Institute (ISI), Kolkata, India. sasanka@isical.ac.in

[1] Here, we assume that the union of lowest-frequency disks covers all the users.

every point is contained in at least one disk and

$$\sum_{i=1}^{n} \min\{1, |\mathsf{small}(p_i)|\}$$

is maximized, where $\mathsf{small}(p_i)$ is the set of selected small disks that contain the point p_i. In other words, we want to select exactly one disk from each object such that all the points are covered and the number of points covered by at least one small disk is maximized.

Notation. For a point p in the plane, we denote the x- and y-coordinates of p by $x(p)$ and $y(p)$, respectively. Moreover, we denote the Euclidean distance between two points p_i and p_j by $\mathsf{dist}(p_i, p_j)$. For an object i, we denote the centres of $\mathsf{sDisk}(i)$ and $\mathsf{bDisk}(i)$ by $\mathsf{sCentre}(i)$ and $\mathsf{bCentre}(i)$, respectively.

Consider an instance of the sDiskCover problem. Let p be an input point that is contained in exactly one big disk $\mathsf{bDisk}(i)$ (for some object i) and not contained in any small disk. Then, any feasible solution must select $\mathsf{bDisk}(i)$. Moreover, let M be the set of all input points $q(\neq p)$ such that (i) q is covered by $\mathsf{bDisk}(i)$ and (ii) no small disk covers q (i.e., q is only covered by big disks). Then, we can include $\mathsf{bDisk}(i)$ into the solution, and then remove the object i and the set $M \cup \{p\}$ from the instance. Therefore, we assume the following throughout the paper.

Assumption 1 *Given an instance of the* sDiskCover *problem, if an input point is not contained in any small disk, then it is contained in at least two big disks.*

2 One-dimensional Objects

In this section, we consider the sDiskCover problem for n points and m one-dimensional objects: each object is a pair of intervals on the real line (i.e., an interval with length one and an interval with length two). For an interval i, we denote its left and right endpoints by $\mathsf{left}(i)$ and $\mathsf{right}(i)$, respectively. Moreover, we write $p(i)$ to denote the set of input points covered by i. For the rest of this section, we refer to the small and big disks of an object i as the small and big *intervals* of i and denote them by $\mathsf{slnt}(i)$ and $\mathsf{blnt}(i)$, respectively (we still use the term "object" whenever we are not referring to a specific interval). Moreover, we assume that the input points have distinct x-coordinates and $x(p_i)$ is distinct from that of the endpoints of any interval in the input objects, for all $1 \leq i \leq n$.

Here, we first consider the sDiskCover problem in a special case in which the objects are *left-aligned*: we have $x(\mathsf{left}(\mathsf{slnt}(i))) = x(\mathsf{left}(\mathsf{blnt}(i)))$ for all objects $1 \leq i \leq m$. In Section 2.2, we will solve the problem without this restriction. In this subsection, we assume that the points are ordered from left to right as $p_1, p_2, \ldots, p_n$, and the objects are sorted from left to right by the x-coordinate of the right endpoint of their *big* interval.

2.1 Left-aligned Intervals

An object i on the real line is called *left-aligned* if $x(\mathsf{left}(\mathsf{slnt}(i))) = x(\mathsf{left}(\mathsf{blnt}(i)))$; a set of one-dimensional objects is called left-aligned if every object in the set is left-aligned. Given a set of n points $p_1, \ldots, p_n$ and m one-dimensional left-aligned objects on the real line, we give an exact $O(mn)$-time algorithm for the sDiskCover problem.

For $1 \leq i \leq n$ and $1 \leq j \leq m$, define $A[i, j]$ to be the objective value of an exact solution for the problem on the points $p_1, p_2, \ldots, p_i$ and the objects $o_1, o_2, \ldots, o_j$. Similarly, define $B[i, j]$ to be the objective value of an exact solution for the problem on the points $p_1, p_2, \ldots, p_i$ and the objects $o_1, o_2, \ldots, o_j$, assuming that $\mathsf{blnt}(o_j)$ is in the solution. Our goal is to compute $A[n, m]$; the actual solution that gives us $A[n, m]$ can be computed in the standard manner. We next show how to compute $A[i, j]$ and $B[i, j]$. First, we need the following lemma.

Lemma 1 *Consider an instance of the* sDiskCover *problem, and let ℓ be the vertical line through $\mathsf{right}(\mathsf{slnt}(o_m))$. Moreover, assume that p_n lies to the right of ℓ, and let T denote the set of all big intervals that intersect ℓ (including $\mathsf{blnt}(o_m)$). Then, there exists an exact solution S for the problem such that $S \cap T \subseteq \{\mathsf{blnt}(o_m), \mathsf{blnt}(o_{m-1})\}$ and $S \cap T \neq \emptyset$.*

Proof. Since p_n lies to the right of ℓ, we have $|T| \geq 2$ by Assumption 1. If $|T| = 2$, then $T = \{o_{m-1}, o_m\}$ and so any feasible solution S must contain at least one of $\mathsf{blnt}(o_{m-1})$ and $\mathsf{blnt}(o_m)$. Hence, $S \cap T \subseteq \{\mathsf{blnt}(o_m), \mathsf{blnt}(o_{m-1})\}$ and $S \cap T \neq \emptyset$.

Now, assume that $|T| > 2$. Consider an exact solution that contains neither $\mathsf{blnt}(o_{m-1})$ nor $\mathsf{blnt}(o_m)$. Then, p_n must be covered by $\mathsf{blnt}(o_i)$ in this solution, for some $i < m - 1$, and so $\mathsf{blnt}(o_i) \in T$. We now replace $\mathsf{blnt}(o_i)$ and $\mathsf{slnt}(o_{m-1})$ with, respectively, $\mathsf{slnt}(o_i)$ and $\mathsf{blnt}(o_{m-1})$ in this solution; let S be the resulting set of intervals. Since the objects are left-aligned, these replacements do not leave any point uncovered: any point that was covered by $\mathsf{blnt}(o_i) \cup \mathsf{slnt}(o_{m-1})$ is still covered by $\mathsf{slnt}(o_i) \cup \mathsf{blnt}(o_{m-1})$. Moreover, since $i < m - 1$ and $o_i \in T$ (i.e., $\mathsf{blnt}(o_i)$ intersects ℓ), any point that was covered by $\mathsf{slnt}(o_{m-1})$ is still covered by $\mathsf{slnt}(o_i) \cup \mathsf{slnt}(o_m)$. Hence, S is a feasible solution for the problem and its objective value is at least as big as that of the initial solution. Observe that $S \cap T \subseteq \{\mathsf{blnt}(o_m), \mathsf{blnt}(o_{m-1})\}$ and $S \cap T \neq \emptyset$. □

Computing $A[i, j]$**.** Let ℓ denote the vertical line through $\mathsf{right}(\mathsf{slnt}(o_j))$. We consider two cases depending on whether p_i lies to the right or to the left of ℓ.

If p_i lies to the right of ℓ, then we can focus our attention to $\mathsf{bInt}(o_j)$ and $\mathsf{bInt}(o_{j-1})$ by Lemma 1. Let ℓ' be the vertical line through $\mathsf{right}(\mathsf{sInt}(o_{j-1}))$ and let $p_{i'}$ be the rightmost point that is to the left of ℓ'. Moreover, let ℓ'' be the vertical line through $\mathsf{left}(\mathsf{sInt}(o_j))$ and let $p_{i''}$ be the rightmost point that is to the left of ℓ''. Now, if $\mathsf{bInt}(o_j)$ is in the solution, then every point to the right of ℓ' is covered by $\mathsf{bInt}(o_j)$ and no such point is contained in a small interval; hence, the problem is reduced to $B[i', j]$. On the other hand, if $\mathsf{bInt}(o_{j-1})$ is in the solution, then we also take $\mathsf{sInt}(o_j)$ into the solution. Hence, the problem is reduced to $B[i'', j-1] + |p(\mathsf{sInt}(o_j))|$. Therefore, we have $A[i, j] = \max\{B[i', j], B[i'', j-1] + |p(\mathsf{sInt}(o_j))|\}$.

If p_i is to the left of ℓ, then we take $\mathsf{sInt}(o_j)$ into the solution. This is because if there exist a solution with $\mathsf{bInt}(o_j)$, then we can replace $\mathsf{bInt}(o_j)$ with $\mathsf{sInt}(o_j)$ without decreasing the objective value. Now, let ℓ' denote the vertical line through $\mathsf{left}(\mathsf{sInt}(o_j))$ and let $p_{i'}$ be the rightmost point that is to the left of ℓ'. Then, the problem is reduced to $A[i', j-1] + p(\mathsf{sInt}(o_j))$. In summary, we compute $A[i, j]$ as follows. First, assume that $i > 1$ and $j > 1$. If p_n is to the right of ℓ, then $A[i, j] = \max\{B[i', j], B[i'', j-1] + |p(\mathsf{sInt}(o_j))|\}$; otherwise, if p_n is to the left of ℓ, then $A[i, j] = A[i', j-1] + p(\mathsf{sInt}(o_j))$. Now, assume that $i = 1$. If p_1 is contained in at least one small interval, then $A[i, j] = 1$; but, if p_1 is contained in no small interval, then $A[i, j] = 0$. Finally, assume that $j = 1$. If there is at least one point that is not contained in $\mathsf{sInt}(o_1)$, then $A[i, j] = 0$; but, if every point is contained in $\mathsf{sInt}(o_1)$, then $A[i, j] = |p(\mathsf{sInt}(o_1))|$.

Computing $B[i, j]$. To compute $B[i, j]$, let ℓ be the vertical line through $\mathsf{right}(\mathsf{sInt}(o_j))$. We again consider two cases. If p_i is to the right of ℓ, then the problem is simply reduced to $B[i-1, j]$. If p_i is to the left of ℓ, then the problem is reduced to $A[i, j-1]$ because we can remove the object o_j from the instance (as we know that $\mathsf{bInt}(o_j)$ has been selected) and then solve the problem with the same points and the objects $o_1, o_2, \ldots, o_{j-1}$. Therefore,

$$B[i, j] = \begin{cases} B[i-1, j], & \text{if } p_i \text{ is to the right of } \ell, \\ A[i, j-1], & \text{if } p_i \text{ is to the left of } \ell. \end{cases}$$

Moreover, to compute the base cases, assume first that $i = 1$. If at least one of $\mathsf{sInt}(o_1), \ldots, \mathsf{sInt}(o_{j-1})$ contains p_1, then $B[i, j] = 1$; otherwise, $B[i, j] = 0$. Now, if $j = 1$, then $B[i, j] = 0$ because we have taken $\mathsf{bInt}(o_1)$ into the solution.

Running time. The tables A and B each have size mn, and we spend $O(1)$ time to fill one entry of A or one entry of B. Hence, the total time spent to fill out A and B is $O(mn)$ and so we have the following theorem.

Theorem 2 *For a set of n points and m left-aligned objects on the real line, the* sDiskCover *problem can be solved in $O(mn)$ time.*

2.2 Arbitrary Intervals

Here, we remove the "left-aligned" restriction and assume that the small interval of an object i can be anywhere within the big interval of the object as long as $\mathsf{dist}(\mathsf{sCentre}(i), \mathsf{bCentre}(i)) \leq 1/2$ (i.e., the small interval is entirely contained in the big interval). In this subsection, we assume that the objects are ordered from left to right by the x-coordinate of the right endpoint of their *small* interval.

Lemma 3 *There exists an optimal solution for the* sDiskCover *problem such that each point is covered by at most two small intervals.*

Proof. Take any optimal solution OPT for the problem. Let $S(p_i)$ denote the set of small intervals in OPT that cover point p_i for all $i = 1, 2, \ldots, n$. For each point p for which $|S(p)| > 2$, we do the following: let o_ℓ (resp., o_r) be the object for which $\mathsf{sInt}(o_\ell) \in S(p)$ (resp., $\mathsf{sInt}(o_r) \in S(p)$) and $x(\mathsf{left}(\mathsf{sInt}(o_\ell)) \geq x(\mathsf{left}(\mathsf{sInt}(o_j)))$ (resp., $x(\mathsf{right}(\mathsf{sInt}(o_r)) \leq x(\mathsf{right}(\mathsf{sInt}(o_j))))$ for all o_j such that $\mathsf{sInt}(o_j) \in S(p)$. Now, for every small interval in $S(p) \setminus \{\mathsf{sInt}(o_\ell), \mathsf{sInt}(o_j)\}$, we replace the small interval in OPT by its big interval. Clearly, every point is covered by at most two small intervals in the resulting set. Moreover, one can verify that the resulting set of intervals will still cover all the points and has the same objective value as OPT. □

We now describe a dynamic programming algorithm. Let $T[i, j]$ denote the objective value of an optimal solution for covering the points $p_1, p_2, \ldots, p_i$ with the objects $o_1, o_2, \ldots, o_j$ (where the latter ordering is by their small interval). Then, the goal is to compute $T[n, m]$. To compute $T[i, j]$, we assume in the following that the union of the j objects cover all the i points (as otherwise we set $T[i, j]$ to -1). Now, take any optimal solution OPT for $T[i, j]$ and let p_r be the rightmost point for which OPT gets a credit; that is, p_r is the rightmost point that is covered by at least one small interval in OPT. Then, by Lemma 3, p_r is covered by either one or two small intervals in OPT. Let us consider these in two cases.

Point p_r is covered by one small interval in OPT. Let o_a be the object such that $\mathsf{sInt}(o_a) \in$ OPT and $\mathsf{sInt}(o_a)$ covers p_r. Let ℓ (resp., ℓ') be the vertical line through $\mathsf{left}(\mathsf{sInt}(o_a))$ (resp., $\mathsf{right}(\mathsf{sInt}(o_a))$). Moreover, let M_a be the set of objects o_t such that $\mathsf{bInt}(o_t)$ intersects ℓ'. To see which interval of the objects in $M_a \setminus \{o_a\}$ are in OPT, take any object $o_t \in M_a \setminus \{o_a\}$. Observe

that if $\mathsf{sInt}(o_t) \in \mathsf{OPT}$, then $\mathsf{sInt}(o_t)$ does not cover any points in $\{p_{r+1}, \ldots, p_n\}$. Moreover, the points that lie to the right of ℓ and to the left of p_r are already covered by $\mathsf{sInt}(o_a)$ (and so for each of which OPT has gained a credit). This means that, the only way OPT could potentially gain points by having $\mathsf{sInt}(o_t)$ is when $\mathsf{left}(\mathsf{sInt}(o_t))$ lies strictly to the left of ℓ. In that case, among all such $\mathsf{sInt}(o_t)$, OPT must have the one with leftmost left endpoint; consider this object and let t^* be the index of its small interval (in the input ordering defined on small intervals). Notice that for all other objects in $M_a \setminus \{o_a\}$, we can have their big intervals in OPT. Let $p_{r'}$ for some $r' \leq r$, be the leftmost point covered by $\mathsf{sInt}(o_{t^*})$. Then, in this case, we have

$$T[i,j] = \max_{\substack{p_r \in \{p_1,\ldots,p_i\} \\ o_a \in \{o_1,\ldots,o_j\}: \\ p_r \in \mathsf{sInt}(o_a)}} \{f(\mathsf{sInt}(o_a), \mathsf{sInt}(o_{t^*})) + T[r'-1, t^*-1]\},$$

where $f(\mathsf{sInt}(o_a), \mathsf{sInt}(o_{t^*}))$ denotes the number of points covered by at least one of $\mathsf{sInt}(o_a)$ and $\mathsf{sInt}(o_{t^*})$.

Point p_r is covered by two small intervals in OPT. Let o_a and o_b be the two objects such that $\mathsf{sInt}(o_a), \mathsf{sInt}(o_b) \in \mathsf{OPT}$ and they both cover p_r. Assume w.l.o.g. that $x(\mathsf{left}(\mathsf{sInt}(o_a))) \leq x(\mathsf{left}(\mathsf{sInt}(o_b)))$; let ℓ (resp., ℓ') be the vertical line through $\mathsf{left}(\mathsf{sInt}(o_a))$ (resp., $\mathsf{right}(\mathsf{sInt}(o_b))$). Let M_{ab} be the set of objects o_t such that $\mathsf{bInt}(o_t)$ intersects ℓ'. To see which interval of the objects in $M_{ab} \setminus \{o_a, o_b\}$ are in OPT, take any object $o_t \in M_{ab} \setminus \{o_a, o_b\}$. Observe that if $\mathsf{sInt}(o_t) \in \mathsf{OPT}$, then $\mathsf{sInt}(o_t)$ does not cover any point in $\{p_{r+1}, \ldots, p_n\}$. Moreover, the points that lie to the right of ℓ and to the left of p_r are already covered by $\mathsf{sInt}(o_a)$ (and so for each of which OPT has gained a point). This means that, if $x(\mathsf{left}(\mathsf{sInt}(o_t))) \geq x(\mathsf{left}(\mathsf{sInt}(o_a)))$, then OPT does not gain any points by having $\mathsf{sInt}(o_t)$. Therefore, the only way OPT could potentially gain points by having $\mathsf{sInt}(o_t)$ is when $\mathsf{left}(\mathsf{sInt}(o_t))$ lies strictly to the left of ℓ. In that case, among all such $\mathsf{sInt}(o_t)$, OPT must have the one with leftmost left endpoint; consider this object and let t^* be the index of its small interval. Notice that for all other objects in $M_{ab} \setminus \{o_a, o_b\}$, we can have their big interval in OPT. Let $p_{r'}$, for some $r' \leq r$, be the leftmost point covered by $\mathsf{sInt}(o_{t^*})$. Then, in this case, we have

$$T[i,j] = \max_{\substack{p_r \in \{p_1,\ldots,p_i\} \\ \{o_a,o_b\} \subseteq \{o_1,\ldots,o_j\}: \\ p_r \in \mathsf{sInt}(o_a) \cap \mathsf{sInt}(o_b)}} \{f(\mathsf{sInt}(o_a), \mathsf{sInt}(o_b), \mathsf{sInt}(o_{t^*})) + T[r'-1, t^*-1]\},$$

where $f(\mathsf{sInt}(o_a), \mathsf{sInt}(o_b), \mathsf{sInt}(o_{t^*}))$ denotes the number of points covered by at least one of $\mathsf{sInt}(o_a), \mathsf{sInt}(o_b)$ and $\mathsf{sInt}(o_{t^*})$. The base case is $T[1,j]$ for all $j = 1, \ldots, m$; we set $T[1,j] = 1$ if p_1 is covered by at least one small interval in $\{\mathsf{sInt}(o_1), \mathsf{sInt}(o_2), \ldots, \mathsf{sInt}(o_j)\}$ and $T[1,j] = 0$, otherwise.

Running time. Given an instance of the problem, we can compute the order of the points and the intervals (as required by the algorithm) in $O(n \log n)$ and $O(m \log m)$ time, respectively. Moreover, within the same time bound, we can preprocess the input to compute the function $f(\mathsf{sInt}(o))$ for all the input objects o. Each entry of the table T can be computed in $O(m^2 n)$ time, and so we have the following theorem.

Theorem 4 *For a set of n points and m arbitrary intervals on the real line, the* sDiskCover *problem can be solved in $O(m^2 n)$ time.*

3 Two-dimensional Objects

In this section, we consider the sDiskCover problem for objects in the plane and show that the problem is NP-hard. Recall that for each object i, we have two disks: $\mathsf{sDisk}(i)$ whose radius is one and $\mathsf{bDisk}(i)$ whose radius is two. Throughout this section, we assume that $\mathsf{sCentre}(i) = \mathsf{bCentre}(i)$; i.e., the disks are centred at the same point.

We show a polynomial-time reduction from Planar Variable Restricted 3SAT (Planar VR3SAT, for short). Planar VR3SAT is a constrained version of 3SAT in which each variable can appear in at most three clauses and the corresponding *variable-clause graph* is planar. Efrat et al. [2] showed that Planar VR3SAT is NP-hard.

Let I_{SAT} be an instance of Planar VR3SAT with K clauses $C_1, C_2, \ldots, C_K$ and N variables $X_1, X_2, \ldots, X_N$; we denote the two literals of a variable X_i by x_i and $\overline{x_i}$. We construct an instance I_{sDC} of our problem such that I_{sDC} has a solution with objective value of at least $MNK + MK/2$, for some M that we will determine its value below, if and only if I_{SAT} is satisfiable. Given I_{SAT}, we first construct the variable-clause graph G of I_{SAT} in the non-crossing comb-shape form of Knuth and Raghunathan [3]. We assume w.l.o.g. that the variable vertices lie on a vertical line and the clause vertices are connected from left or right of that line; see Figure 2 (left) for an example. This representation has size polynomial in N and K.

Gadgets. For each variable $X_i \in I_{\mathsf{SAT}}$, we replace the corresponding variable vertex in G with two objects as shown in Figure 2 (right); we call this pair of objects the *variable gadget* of X_i. The top object serves as literal $\overline{x_i}$ while the bottom object serves as literal x_i. The variable gadget initially contains three disjoint *group of points*; we call each such group of points a *cloud*. There is one cloud of K points that is shared between $\mathsf{bDisk}(x_i)$ and $\mathsf{bDisk}(\overline{x_i})$, called a *variable-shared cloud*. Moreover,

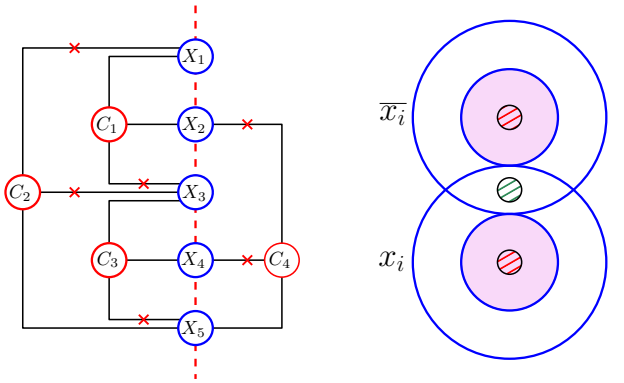


Figure 2: **Left**: an instance of the Planar VR3SAT problem in the comb-shape form of Knuth and Raghunathan [3]. Crosses on the edges indicate negations; for example, $C_2 = (\overline{x_1} \vee \overline{x_3} \vee x_5)$. **Right**: A variable gadget. The three clouds of the gadget are shown as small shaded disks.

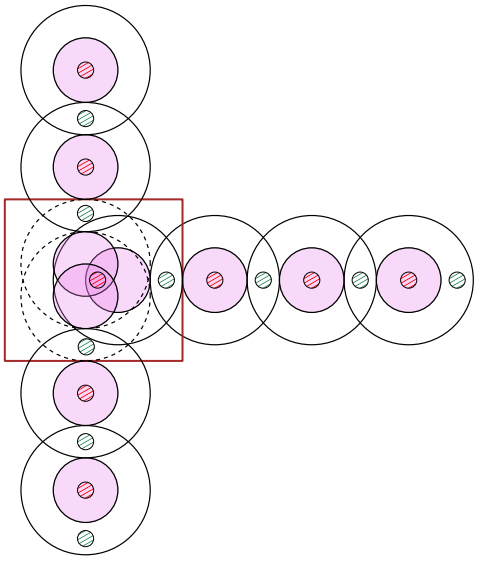


Figure 3: An illustration of a clause gadget.

each of $\mathsf{sDisk}(x_i)$ and $\mathsf{sDisk}(\overline{x_i})$ contains one cloud, each of which we refer to as a *variable-small cloud*. We determine the number of points in a variable-small cloud later.

Observation 1 *Given any feasible solution S for the* sDiskCover *problem, at most one of* $\mathsf{sDisk}(x_i)$ *and* $\mathsf{sDisk}(\overline{x_i})$ *can be in S, for any variable X_i.*

The idea behind the variable gadget (corresponding to a variable X_i) is to ensure that also at most one of $\mathsf{bDisk}(x_i)$ and $\mathsf{bDisk}(\overline{x_i})$ appears in any feasible solution. Then, we set the variable to true if and only if $\mathsf{bDisk}(x_i)$ is in the solution. However, the gadget as it is now does not enforce this. To enforce this, we must enforce one of the small disks to be selected in any feasible solution (hence, forcing its big disk not to be selected). To this end, we will set the number of points in each variable-small cloud (in the variable gadget) to a large enough value that any feasible solution must contain at least one small disk from every variable gadget in order for its objective value to meet a minimum requirement. We will determine this minimum requirement later.

If the literal x_i (resp., $\overline{x_i}$) appears in a clause, then the bottom object (resp., top object) of the gadget is connected to the corresponding clause by a chain of objects, called a *wire*. A wire starting from the bottom object (resp., top object) of a variable gadget and ending at a clause has the following structure. (i) Every object i in the wire has a cloud of K points in $\mathsf{sDisk}(i)$. (ii) The big disk of the first object of the wire shares one cloud of K points with the big disk of the bottom object (resp., top object) in the variable gadget. (iii) The big disks of every two consecutive objects in the wire share one cloud of K points. We call the first object of a wire (that shares a cloud with one of the objects in the variable gadget) the *starting object* of the wire.

For the clause gadget, where three wires meet, the small disks of the last three objects (each arriving from one of the wires) will have a non-empty intersection in which we place one cloud containing K points; see Figure 3. We call this K points a *clause cloud*.

Construction details. Let S be a feasible solution for the problem, and consider the object x_i shown in Figure 2 (right). The variable-small cloud of this object is covered by either $\mathsf{sDisk}(x_i)$ or $\mathsf{bDisk}(x_i)$ in S. If it is covered by $\mathsf{sDisk}(x_i)$ in S, then we must cover the other clouds in this object by the starting objects of the wires connected to x_i. But, if it is covered by $\mathsf{bDisk}(x_i)$ in S, then we can select the small disk of the starting object of each wire. Consequently, depending on whether $\mathsf{sDisk}(x_i)$ or $\mathsf{bDisk}(x_i)$ is selected, one can see that the clouds in the wires connected to this object are covered in S in one of the two possible ways. (Here, we are assuming that S needs to meet the minimum requirement for its objective value.) By re-scaling and adjusting the length of a wire, we can ensure that exactly one of these two possible coverings will let S to select the small disk of the last object in the wire; the other will only let S to select the big disk of the last object. In other words, exactly one of these two possible ways allows S to gain K points for covering the corresponding clause cloud.

By the discussion above, we set a variable X_i to true if and only if $\mathsf{bDisk}(x_i)$ is selected. By having an appropriate number of objects in each wire (while keeping the size polynomial), we can assume that S gains K points for covering the clause cloud (i.e., the small disk of the last object in the wire is selected) if and only if $\mathsf{bDisk}(x_i)$ is selected (i.e., variable X_i is set to true). Notice that

selecting $\mathsf{bDisk}(x_i)$ forces S to select $\mathsf{sDisk}(\overline{x_i})$. Now, by adjusting the number of objects in a wire connecting $\overline{x_i}$ to a clause gadget, we also ensure that the big disk of the last object of the wire is selected. That is, $\mathsf{sDisk}(\overline{x_i})$ is selected if and only if the big disk of the last object of the corresponding wire is selected (i.e., S does not gain any points from the corresponding clause cloud). Finally, we require by re-scaling that the number of objects in each wire is even. This concludes the consistency for the truth assignment of X_i.

We first prove that the number of objects in each wire is polynomial in N and K. Consider an edge in the graph and let L be its length. Notice that since the drawing is polynomial in N and K, so is L. Moreover, this edge can have either no bends or one bend. Our goal is to have each wire consistent with the drawing of its corresponding edge in G; hence, making each wire having no bends or one bend. Suppose first that the edge has no bends. Since the distance between every two consecutive centres of the disks in the wire is three, we have at most $\lfloor L/3 \rfloor$ objects in the wire. Now, suppose that the edge has one bend and let L_1 and L_2 denote the lengths of its segments (i.e., $L = L_1 + L_2$). Then, by a similar argument, the corresponding wire will have at most $\lfloor L_1/3 \rfloor + \lfloor L_2/3 \rfloor$ objects. We therefore conclude that the number of objects in both cases is polynomial in N and K.

In the full version of the paper, we prove that the wires can be connected to variable gadgets such that the objects from different wires do not intersect each other (except at clause gadgets and/or slightly at variable gadgets). To ensure this, we might require to "reroute" some of the wires; hence, making new bends. However, one can verify that the number of objects in each wire remains polynomial in N and K. The proof of the following lemma is given in the full version of the paper.

Lemma 5 *Let S be a feasible solution for the problem with objective value of at least $MNK + MK/2$. Then, S has exactly one big disk and exactly one small disk from every variable gadget.*

Lemma 5 gives us the minimum objective value for a feasible solution that we will use to argue that then the instance I_{SAT} is satisfiable.

Lemma 6 *There exists a feasible solution S for I_{sDC} with objective value of at least $MNK + MK/2$ if and only if I_{SAT} is satisfiable.*

Proof. ($\Rightarrow$) Let S be a feasible solution for I_{sDC} with objective value of at least $MNK+MK/2$. By Lemma 5, we know that there is exactly one big disk from every variable gadget in S. For each variable X_i, $1 \leq i \leq N$, we set the variable to true if and only if $\mathsf{bDisk}(x_i)$ is in S; otherwise, we set X_i to false. To show that this results in a truth assignment, suppose for a contradiction that there exists a clause C that is not satisfied by this assignment. Take any variable $X \in C$. If $x \in C$, then the variable X is set to false (resp., true) by the assignment. This means that S has selected the small disk of object x. Consequently, the big disk of the last object in the wire connecting C to x_i is selected by S: the solution S did not gain K points from the cloud of C. Analogously, if $\overline{x} \in C$, then the variable X is set to true by the assignment. This means that S has selected $\mathsf{sDisk}(\overline{x_i})$. Consequently, the big disk of the last object in the wire connecting C to $\overline{x_i}$ is selected by S: again, the solution S did not gain K points from the cloud of C. Therefore, S cannot have an objective value of $MNK + MK/2$—a contradiction.

($\Leftarrow$) Given a truth assignment for I_{SAT}, we construct a feasible solution S for I_{sDC} with objective value $MNK + MK/2$ as follows. For each variable X_i in I_{SAT}, where $1 \leq i \leq N$: if X_i is set to true, then we add $\mathsf{bDisk}(x_i)$ to S; otherwise, we add $\mathsf{sDisk}(x_i)$ to S. This selection ensures that we get MK points from each variable gadget. Moreover, by selecting the corresponding disk of x_i, we will select the disks in the wires connected to x_i accordingly by alternating between small and big disks. The same also happens for the wires that are connected to $\overline{x_i}$. One can consequently argue that, within a wire, exactly half of the small disks are selected; that is, we will gain $MK/2$ points by covering these clouds using small disks. Therefore, S has the objective value $MNK + MK/2$. □

By Lemma 6, we have the following theorem.

Theorem 7 *The* sDiskCover *problem is* NP*-hard for concentric disks in the plane.*

References

[1] D. Bhaumick and S. C. Ghosh. Efficient multicast association to improve the throughput in IEEE 802.11 WLAN. *MONET*, 21(3):436–452, 2016.

[2] A. Efrat, C. Erten, and S. G. Kobourov. Fixed-location circular arc drawing of planar graphs. *J. Graph Algorithms Appl.*, 11(1):145–164, 2007.

[3] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. Discrete Math.*, 5(3):422–427, 1992.

Hardness of Approximation for Red-Blue Covering

Sima Hajiaghaei Shanjani*

Abstract

In Red-Blue Geometric Set Cover a set of red points, a set of blue points, and a set of objects are given and the goal is to find a subset of the objects that cover all the blue points while covering the minimum number of red points. Chan and Hu in 2014 showed that the problem is NP-hard even when the points are in the plane and objects are axis-aligned unit squares. Here we study Red-Blue Geometric Set Cover when the objects are axis-aligned rectangles, convex shapes, and triangles. We also study the problem of Boxes Class Cover: a red-blue point set is given, and the goal is to find a minimum number of axis-aligned rectangles that cover all the blue points but no red. This problem is introduced in a paper in 2012 by Bereg et al., who showed the problem is NP-hard.

We prove the following: 1) Red-Blue Geometric Set Cover is APX-hard when the objects are axis-aligned rectangles. 2) Red-Blue Geometric Set Cover cannot be approximated to within $2^{\log^{1-1/(\log\log m)^c} m}$ in polynomial time for any constant $c < 1/2$, unless $P = NP$, when the given objects are m triangles or convex objects. 3) Boxes Class Cover is APX-hard.

In the non-geometric setting Red-Blue Set Cover is known to be strictly harder to approximate than Set Cover. In the geometric setting, no such a relation between Geometric Set Cover and Red-Blue Geometric Set Cover was previously known. We show that there is a class of objects, triangles, for which approximating Red-Blue Geometric Set Cover is strictly harder than approximating Geometric Set Cover.

We also define a restricted version of Max3SAT, MaxRM-3SAT, and we prove that this problem is APX-hard. This problem might be interesting in its own right.

1 Introduction

Geometric Set Cover, the geometric version of Set Cover, is a fundamental theoretical problem that has been studied for over 30 years. Applications of this problem include wireless network design, image compression, and circuit-printing [10]. This problem is NP-hard even for simple geometric objects such as unit squares, unit disks, and axis-aligned rectangles [19]. Much effort has been made to develop approximation algorithms or prove lower bounds for the best possible approximation ratio for this problem. While Geometric Set Cover has been studied widely for several geometric objects, the problem of Red-Blue Geometric Set Cover has been studied only for axis-aligned unit squares [11].

In this paper, we study Red-Blue Geometric Set Cover for some classes of objects, and we prove new results on the hardness of approximation for this and Boxes Class Cover. Then we compare our result with the approximability of Geometric Set Cover for the same class of objects, and we conclude that Red-Blue Geometric Set Cover is a harder problem than Geometric Set Cover for some class of objects. In the following section, we define and describe related works to these problems.

1.1 Problems, and Related Works

We recall that for an optimization problem, a polynomial-time approximation scheme (PTAS) is a $(1+\epsilon)$-approximation algorithm which takes a parameter $\epsilon > 0$ as part of the input and is polynomial in the problem size n for every fixed ϵ. An optimization problem is APX-hard if no PTAS exists for the problem unless $P = NP$.

***Set Cover*:** A universe set X of n elements and a family T of m subsets of X are given, and the goal is to find a minimum sized subset $T' \subseteq T$ such that each element in X is contained in at least one member of T'. This fundamental problem has been known to be NP-hard and NP-hard to approximate within a factor of $(1-\alpha)\ln n$ of the optimum for every $\alpha > 0$ [14]. It has also been shown that this problem cannot be approximated to within $2^{\log^{1-\delta_c(m)} m}$ in polynomial time for any constant $c < 1/2$ unless SAT can be decided in time $2^{O(2^{\log^{1-\delta_c(n)} n})}$, where $\delta_c(n) = 1/(\log\log n)^c$ [23].

Geometric Set Cover(GSC): In this version of Set Cover, we are given a set of points X and a family T of geometric objects, and the goal is to find a minimum sized subset $T' \subseteq T$ such that each point is covered with at least one of the selected objects. The results of this problem can be studied in two directions.

First, PTASes have been developed for some simple objects, such as unit-squares [19] and disks in $\mathbb{R}^2$ [22]. For some other objects with low VC-dimension, ϵ-net based

*Department of Computer Science, University of Victoria, sima@uvic.ca, [Research funded by NSERC Discovery Grant RGPIN 2016-04234]

algorithms with constant or almost-constant approximation factors have been presented [7, 15]. However, finding a small, $O(1/\epsilon)$-size , ϵ-net is not always possible. For example, [24] shows that there exist a dual range space induced by a family of finite families of axis-aligned rectangles in which the size of the smallest ϵ-net is $\Omega(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$. In this case the approximation factor of the ϵ-net based algorithms is $O(\log OPT)$, where OPT is the size of the optimal solution. Second, it has been shown that Geometric Set Cover is APX-hard for a large class of geometric objects including axis-aligned rectangles, axis-aligned slabs, and triangles[10][17].

In this paper we denote the problem of Geometric Set Cover when the objects are from the class of objects OBJ with GSC[OBJ], e.g. GSC[AARectangle].

Class Cover:In class cover problems, points are given in two sets R and B, red points and blue points respectively, and the goal is to find a minimum sized family T of a specific type of objects (e.g. balls) that cover all the points in B but no point in R. Note that in Set Cover problems, the family of subsets is given as the input, and we select a subfamily of them. But, in class cover, the goal is to compute such a family and the candidate subsets are all the subsets of that type of object. This problem has been studied also in the context of data mining when the objects are balls with the constraint that balls are centered at blue points [8, 13, 20]. The constraint and non-constraint version of this problem is NP-hard when the objects are balls [8, 2]

Boxes Class Cover(BCC): In this version of Class Cover problem, the objects are axis-aligned rectangles. Bereg et.al. use a reduction from a rectilinear polygon covering problem [12, 21] to show that the BCC is NP-hard and admits $O(\log OPT)$-approximation, where OPT is the size of optimal covering [3]. The same paper also shows NP-hardness and the existence of an $O(1)$-approximation algorithm for BCC when the objects are axis-aligned squares. They also show that BCC is NP-hard even if the objects are axis-aligned half-slabs, but BCC can be exactly solved in polynomial time when the objects are axis-aligned slabs. However, no hardness of approximation has been shown for BCC yet. [1] also shows that BCC is NP-hard by a reduction from a version of Max3SAT, NAS-SAT, to this problem.

Note that BCC is a restricted version of GSC[AARectangle]. In BCC a family of rectangles is not given, but we can consider the family of all the possible eligible rectangles as the family of given objects for the Geometric Set Cover. It is interesting to see that Geometric Set Cover is strictly harder than BCC. This comes from the fact that Geometric Set Cover is APX-hard, but BCC can be solved in polynomial time exactly when the objects are axis-aligned slabs.

Red-Blue Set Cover: This is a more general version of Set Cover, where the elements are given in two sets R and B, red elements and blue elements, and the goal is to select a subfamily T' of a given family T of subsets of $R \cup B$ such that T' covers all the elements in B, but includes only the minimum number of elements in R. Carr et.al. in [9]showed that the problem is NP-hard and NP-hard to approximate within $2^{(\log m)^{(1-\delta)}}$ factor of the optimal for $\delta = 1/\log^{\alpha}\log m$ and any constant $\alpha < 1/2$ even in the restricted case that each set in T contains only one blue and two red elements. They also present a $2\sqrt{m}$-approximation algorithm for the case that each set in T contains only one blue element.

Red-Blue Geometric Set Cover (RBGSC): In the geometric version of Red-Blue Set Cover, the given sets R and B are points and the given family T are geometric objects. Chan and Hu showed that the problem is NP-hard even when the objects are unit-squares and present a PTAS for this version of the problem [11]. To the best of our knowledge, no hardness of approximation result has been shown for Red-Blue Geometric Set Cover.

Note that Red-Blue Set cover is a general version of Set Cover: any instance of Set Cover can be transformed to an instance of Red-Blue set Cover by considering all the elements of X as blue elements and adding exactly one distinct red element to each subset in T. However, this reduction does not work for the geometric version. This is because it is not always possible to add exactly one distinct red point to each member of T. So, Red-Blue Geometric Set Cover is not yet shown to be a more general version of Geometric Set Cover, and we do not know a way to relate lower bounds on these two geometric problems. We independently show a hardness of approximation for the Red-Blue Geometric Set Cover.

In this paper, we denote the problem of Red-Blue Geometric Set Cover when the objects are from the class of objects OBJ with RBGSC[OBJ]. e.g. RBGSC[AARectangle].

We show APX-hardness of RBGSC[AARectangle] and Boxes Class Cover via reductions from a newly defined version of Max3SAT. We also show how we can modify reductions in [11] and [3] to prove the APX-harness of these two problems.

Max3SAT: This is the version of MaxSAT where a CNF formula is given and each clause has at most 3 distinct literals, and the goal is to determine the maximum number of clauses that can be satisfied by any assignment. Håstad showed that MaxE3SAT, the version of Max3SAT in which each clause is size of exactly three, is NP-hard to approximate within a factor greater than 7/8 of the optimum even in the case of satisfiable instances of the problem [18]. Here we use MAX-EkSAT-b, a version of MaxSAT in which every clause has length k and each variable occurs exactly b times (other notations have been used for this problem e.g. (k, b)-SAT , EbOCC-EkSAT, and MAX EkSAT(b) [5]).

Feige showed that MaxE3SAT-5 is hard to approximate within a specific constant number [16]. MaxE3SAT-4 is also shown to be hard to approximate within a specific constant factor of the optimal [5] [6].

1.2 Our Contribution

We present proof of hardness of approximation for some geometric problems listed below:

- RBGSC[Axis-Aligned Rectangles] is APX-hard.
- RBGSC[Triangle] and RBGSC[Convex] cannot be approximated to within $2^{\log^{1-1/(\log\log m)^c} m}$ in polynomial time for any constant $c < 1/2$, unless $P = NP$, where m is the number of given triangles or convex objects.
- Boxes Class Cover for axis-aligned rectangles is APX-hard.

These results show that there is a class of objects, triangles, for which approximating Red-Blue Geometric Set Cover is strictly harder than approximating Geometric Set Cover.

we also obtain to define a new version of Max3SAT, MaxRM-3SAT, in Definition 1, and then prove that this problem is also APX-hard.

1.3 Outline of the Paper

We show our hardness results by a series of reductions. Figure 1 shows these reductions. In section 2 we define a new version of Max3SAT, MAX Restricted Mixed 3SAT (MaxRM-3SAT), and we show that the problem is APX-hard. In Section 2 we only provide the idea of the reduction, and the details of this reduction appear in Appendix A. In Section 3 we prove the APX-hardness of RBGSC[AARectangle] in two ways: First, we show a reduction from MaxRM-3SAT to RBGSC[AARectangle]. Second, we show how to modify the presented reductions in [10] to show that RBGSC[AARectangle] is APX-hard. As the reduction from MaxRM-3SAT to BCC is similar to the reduction we described in Section 3, we provide the proof of APX-hardness of BCC in Appendix B. In Appendix B, we also mention how we can use the reduction presented in [3] to show the similar result. In Section 4 we prove hardness results for RBGSC[Convex] and RBGSC[Triangle] by reductions from Set Cover and Red-Blue Set Cover.

2 MaxRM-3SAT

Max3SAT is a version of MaxSAT where all clauses have at most 3 literals and the goal is to determine the maximum number of clauses that can be satisfied by any assignment. Here, we define MaxRM-3SAT, and we prove this problem is APX-hard.

Figure 1: Reductions. * This is a modified version of the reduction that Chan and Grant showed from SPECIAL-3SC to GSC[AARectangle] in [10]. ** This is the reduction that Bereg et.al. showed from Rectilinear Polygon Covering to BCC in [3].

Definition 1 (MaxRM-3SAT) *This problem is a variant of Max3SAT where all the clauses are of size 2 or 3 and have the following properties:*

1. *All the clauses of size 3 have a literal in negated form and a literal in non-negated form.*
2. *Any variable appears in exactly one clause of size 3, i.e., if v_i is a variable in this formula, only one of v_i or $\bar{v_i}$ can appear in any clause of size 3.*
3. *Any variable appears in exactly one of the clauses of size 2 in negated form, and exactly one of the clauses of size 2 in non-negated form.*

We can observe that by properties 2 and 3 of the definition if m is the number of clauses of size 3 in an instance of MaxRM-3SAT, then there are exactly $3m$ variables and $4m$ clauses in total in the formula.

We prove a hardness result for MaxRM-3SAT by a reduction from MaxE3SAT-5, the version of the Max3SAT in which each clause is of length exactly 3 and each variable appears in exactly 5 clauses [16].

Theorem 2 *MaxRM-3SAT problem is NP-hard to approximate within specific constant factor C_{RM-SAT} of the optimum. (Proof in Appendix A.)*

3 RBGSC[AARectangle]

RBGSC[AARectangle]: *A set of red points R, a set of blue points B, and a family of axis-aligned rectangles T are given, the goal is to select a subfamily T' of a given family T such that T' covers all the elements in B, but includes the minimum number of elements in R.*

The hardness result that we prove for RBGSC[Triangle] in this section shows that Red-Blue Geometric Set Cover is APX-hard, same as Geometric Set Cover, when the objects are axis-aligned rectangles.

3.1 Reduction from MaxRM-3SAT to RBGSC[AARectangle]

In this Section, for an instance of MaxRM-3SAT we construct a set of red points, a set of blue points, and a set of rectangles in polynomial time. Then, we show a relation between the number of satisfied clauses in an optimal solution of MaxRM-3SAT and the size of the optimum solution in the corresponding instance of RBGSC[AARectangle]. The idea of this structure was inspired by the structure used in [1].

For Φ, an instance of MaxRM-3SAT with $3m$ variables and $4m$ clauses, we change the order of the clauses to have all the clauses of size 3 first and then clauses of size 2. We rename the jth variable of the kth clause of this order to $X_{3(k-1)+j}$.

For each variable X_i, $1 \leq i \leq 3m$, we add 4 blue points to set B and two *vertical* and two *horizontal* rectangles to the object set T as shown in Figure2(a). These axis-aligned rectangles for each variable only cover blue points associated with their variable. On this arrangement of points RBGSC[AARectangle] has to have an optimal solution that covers each variable's blue points by exactly two rectangles, either both *vertical* or both *horizontal*. We call these blue points that we added to B *variable points*, and the rectangles *variable rectangles*.

The main idea of the reduction from MaxRM-3SAT to BCC is that the choice of vertical vs horizontal corresponds to a true vs a false assignment to the variables. For each clause, we add some blue points to B to force the choice of the covering rectangle to be *horizontal* or *vertical* in the optimal solution for RBGSC[AARectangle] based on the structure of the clauses of Φ. The locations of these points are different in each type of clauses depending on the size of the clause and the number of negated literals in the clause. Figure 2 (c), (d), (e), and (f), show these new blue points added to B and the associated rectangles added to T. We call these added blue points *clause points* and the rectangles *clause rectangles*.

Finally, we add one distinct red point to each rectangle. Figure2 illustrates that there is a region in each rectangle that does not overlap with the other rectangles.

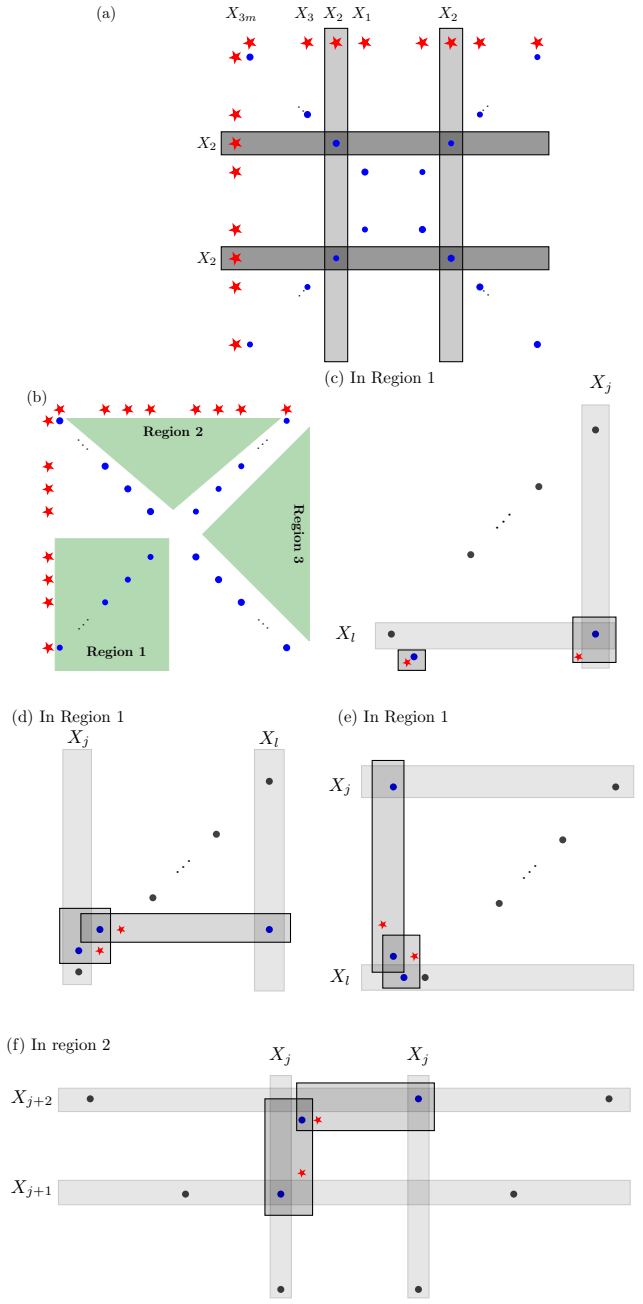


Figure 2: a) *variable points* for all the variables and four *variable rectangles* for X_2 (b)Highlighted green areas are divisions of the plane to Region 1-3. *clause points* and *clause rectangles* for different types of clauses: c) $c = (X_j \vee \bar{X}_l)$, d) $c = (X_j \vee X_l)$, e) $c = (\bar{X}_j \vee \bar{X}_l)$, f) $c = (X_j \vee \bar{X}_{j+1} \vee \bar{X}_{j+2})$ (For $c = (\bar{X}_j \vee X_{j+1} \vee X_{j+2})$, added points are similar to part (e) but rotated by $-\pi/2$ in Region 3.) In the figures of this paper, circles and stars indicate blue and red points respectively. *Clause points* are shown in blue and the *variable points* are shown in black.

Observation 1 *a) For each clause c and an assignment for Φ, if the clause is satisfied, then one extra rectangle in addition to 'variable rectangles' is needed to cover clause points of c; otherwise two extra rectangles are needed to cover clause points of c. b) $10m$ rectangles are needed to cover blue points in B.*

Lemma 3 *If there is an assignment for Φ with $4m - k$ satisfied clauses, then there is a solution for the corresponding instance of RBGSC[AARectangle] with at most $10m + k$ rectangles.*

Proof. The following mapping $\mathcal{G}$ maps an assignment α for Φ to a solution T' for RBGSC[AARectangle]: For any variable X_i, if $\alpha(X_i)$ is true, then add two *hori-*

zontal variable rectangles associated with X_i to T'. For any variable X_i, if $\alpha(X_i)$ is false, then add two *vertical variable rectangles* associated with X_i to T'. For each satisfied clause c, add one of the *clasue rectangles* associated with c to T'. For each unsatisfied clause c, add both of the *clasue rectangles* associated with c to T'.

The rectangles in T' cover all the blue points and the size of T' is $10m + k$. □

Lemma 4 *If there is a solution with $10m+k$ rectangles for this instance of RBGSC[AARectangle], then there is an assignment for Φ with at least $4m - k$ satisfied clauses.*

Proof. Let T, $|T| = 10m + k$, denote the set of rectangles in this solution for RBGSC[AARectangle]. Here we show how we can define a mapping $\mathcal{G}^{-1}$ that maps a solution T for RBGSC[AARectangle] to an assignment α for Φ.

Assume all the four *variable points* of any variable is covered with exactly two *vertical* or exactly two *horizontal* rectangles in T, then $\mathcal{G}^{-1}$ is defined as below. If T does not have such a property, later we show how we can find another solution T' that satisfies this property with the same or lower number of rectangles than T.

$\mathcal{G}^{-1}$: $\alpha(X_i) = 1$; if the *variable points* of X_i are covered with *vertical* rectangles in T'. $\alpha(X_i) = 0$; if the *variable points* of X_i are covered with *horizontal* rectangles in T'.

Here we show there is the solution T', in which all the four *variable points* of any variable is covered with exactly two *vertical* or exactly two *horizontal* rectangles and $|T| \geq |T'|$. Observe that for any variable, each *variable point* can be covered only by its *variable rectangles*. Set $T' = \emptyset$. For any variable X_i, $1 \leq i \leq 3m$, if T covers the variable points of X_i with exactly two rectangles, this means either both of them are *vertical* or both of them are *horizontal*. In this case, we add both of these rectangles to T'. In the case that T covers the variable points of X_i with three or four rectangles, check if the two *vertical* or the two *horizontal* rectangles cover the most number of *clause points* of the clauses that X_i appears in. Add these two *variable rectangles* to T'. If a *clause point* remains uncovered, add one *rectangle clause* to T' to cover that. Note that no more than one *clause point* might remain uncovered as by the definition of MaxRM-3SAT each variable only appears in three clauses, two clauses of size 2 and one clause of size 3, so either the two *vertical* or the two *horizontal* rectangles cover at least two of these clauses. Therefore, for this case, we added three or lees rectangles to T'. □

In this instance of RBGSC[AARectangles], each object covers a distinct red point, so minimizing the number of covered red points also minimizes the number of objects.

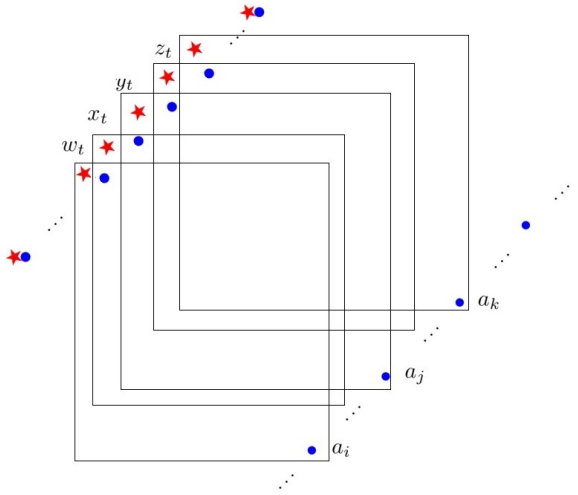


Figure 3: Points and rectangles for Reduction from SPECIAL-3SC to RBGSC[AARectangle]

3.2 Reduction from SPECIAL-3SC to RBGSC[AARectangle]

In this section, we show how we can modify the presented reduction in [10] from SPECIAL-3CS, a restricted version of Set Cover, to GSC[AARectangle] to show that RBGSC[AARectangle] is APX-hard.

ESPECIAL-3SC [10]: we are given universe set $U = A \cup W \cup X \cup Y \cup Z$ comprising disjoint sets $A = \{a_1, ..., a_n\}$, $W = \{w_1, ..., w_m\}$, $X = \{x_1, ..., x_m\}$, $Y = \{y_1, ..., y_m\}$, and $Z = \{z_1, ..., z_m\}$ where $2n = 3m$. We are also given a family S of $5m$ subsets of U satisfying the following two conditions: 1) for each $a \leq t \leq m$, there are integers $1 \leq i < j < k \leq n$ such that S contains the sets $\{a_i, w_t\}$, $\{w_t, x_t\}$, $\{a_j, x_t, y_t\}$, $\{y_t, z_t\}$, and $\{a_k, z_t\}$ (summing over all t given the $5m$sets contained in S.) 2) for all $1 \leq t \leq n$, the element a_t is in exactly two sets in S. SPECIAL-3SC denotes the Set Cover on universe set U and subset set S.

Chan and Grant showed SPECIAL-3SC is APX-hard by a reduction from minimum vertex cover on 3-regular graphs [10]. Then, they showed a reduction from SPECIAL-3SC to GSC[Fat Axis-Aligned Rectangles]: place the elements of A, in order, on the line segment $\{(x, x-2) : x \in [1, 1+\epsilon]\}$ and place the elements of $A' = W \cup X \cup Y \cup Z$, in order, on the line segment $\{(x, x+2) : x \in [-1, -1+\epsilon]\}$, for a sufficiently small $\epsilon > 0$. Then, add $5m$ axis-aligned rectangles covering $\{a_i, w_t\}$, $\{w_t, x_t\}$, $\{a_j, x_t, y_t\}$, $\{y_t, z_t\}$, and $\{a_k, z_t\}$ for any $1 \leq t \leq m$. Figure 1.(C1) of [10] shows these points and rectangles.

The following is the modified reduction for RBGSC[AARectangle].

Reduction from SPECIAL-3SC to RBGSC[AARectangle]: Add m blue points on the line $y = x - 2$ for the elements in A, and add $5m$

blue points on the line $y = x + 2$ for the elements in $A' = W \cup X \cup Y \cup Z$. For each set $s_i \in S$ add the axis-aligned rectangles shown in Figure 3. Note that here the rectangles are slightly different than the rectangles in [10]. This is to make sure that there is an area in each rectangle which is not covered by any other rectangle. Therefore, we can add a distinct red point to each rectangle. Thus, an optimal solution for RBGSC[AARectangle] that minimizes the number of covered red points also minimizes the number of rectangles and so an optimal solution for SPECIAL-3SC.

4 RBGSC[Convex] and RBGSC[Triangle]

Theorem 5 *For every $\alpha > 0$ it is NP-hard to approximate RBGSC[AARectangle] within $(1-\alpha)\ln b$ of the optimum, where b is the number of blue points.*

Proof. Suppose that we have an instance of Set Cover, in which X is the set of n elements and T is a family of m subsets of X. The following transformation ψ, transforms this instance of Set Cover to an instance of RBGSC[Convex], where R is the set of red points, B is the set of blue points, and O is the set of convex objects.

ψ takes an arbitrary circle on the plane, and for each $x_i \in X$, ψ adds the blue point b_i on the circle. For each $s_i \in T$, ψ adds the red point r_i on the circle as shown in Figure 4. For set of objects O, for each $s_i \in T$, ψ adds the convex shape o_i, which is defined by connecting r_i and the blue points corresponded to s_i's members, i.e. $o_i = ConvexHull(\{r_i\} \cup \{b_j | x_j \in s_i\})$.

In this instance of RGBSC[Convex], each object covers exactly one distinct red point. So, any solution for set cover with size k gives a solution for this instance of RBGSC[Convex] in which k red points are covered. Besides, for any solution of RBGSC[Convex] with k covered red points, there is a solution for set cover with size k. □

Theorem 6 *Red-Blue Geometric Set Cover is cannot be approximated to within $2^{\log^{1-1/(\log\log m)^c} m}$ in polynomial time for any constant $c < 1/2$, unless $P = NP$, where the objects are convex objects and m is the number of given convex objects.*

Proof. The proof of this theorem is similar to Theorem 5. If X_B, X_R, and T are the inputs of an instance of Red-Blue Set Cover, for each $xb_i \in X_B$,add the blue point b_i, on the circle, to B similar to Figure 4. For each $xr_i \in X_R$, add the red point r_i, on the circle, to R similar to Figure 4. For each $s_i \in T$, add convex shape o_i, which is defined by connecting red points and blue points corresponded to s_i's members, i.e. $o_i = ConvexHull(\{b_j | xb_j \in s_i\} \cup \{r_l | xr_l \in s_i\})$. This implies that RBGSC[Convex] is as hard as Red-Blue Set Cover, which has been shopwn to be NP-hard to approximate within $2^{(\log m)^{(1-\delta)}}$ factor of the optimal for $\delta = 1/\log^{\alpha}\log m$ and any constant $\alpha < 1/2$ [9]. □

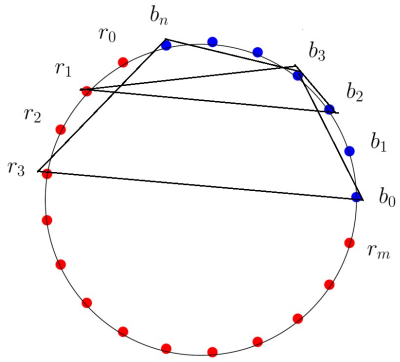


Figure 4: points and convex shapes for the reduction from Set Cover to RBGSC[Convex]

Carr et.al. showed that their hardness result holds even in the restricted case that each set in T contains only one blue and two red elements [9]. Here we reduce this version of the Red-Blue Set Cover to RBGSC[Triangle].

Theorem 7 *Red-Blue Geometric Set Cover is cannot be approximated to within $2^{\log^{1-1/(\log\log m)^c} m}$ in polynomial time for any constant $c < 1/2$, unless $P = NP$, where the objects are triangles and m is the number of given triangles.*

Proof. This proof is also similar to is similar to the proof of Theorem 5. Assume X_B, X_R, and T are the inputs of an instance of Red-Blue Set Cover, where each $s_i \in T$ contains only one blue and two red elements. For each $xb_i \in X_B$,add the blue point b_i, on the circle, to B similar to Figure 4. For each $xr_i \in X_R$, add the red point r_i, on the circle, to R similar to Figure 4. For each $s_i \in T$, add triangle t_i, which is defined by connecting the two red points and the one blue point corresponded to s_i's members, i.e. $t_i = Triangle(\{b_i | xb_i \in s_i\} \cup \{r_l | xr_l \in s_i\})$.

This implies that RBGSC[Triangle] is as hard as the restricted version of Red-Blue Set Cover, which has been shopwn to be NP-hard to approximate within $2^{(\log m)^{(1-\delta)}}$ factor of the optimal for $\delta = 1/(\log\log m)^{\alpha}$ and any constant $\alpha < 1/2$ [9]. □

The hardness result in Theorem 7 shows that Red-Blue Geometric Set Cover is strictly harder than Geometric Set Cover, when the objects are triangles. This is because the VC-dimension of triangles in the plane is 7, so the approximation factor of ϵ-net based algorithms on GSC[Triangle] is $O(\log OPT)$, which is smaller than the lower bound we showed for RBGSC[Triangle] in Theorem 7.

References

[1] N. Assadian. Separating colored points. M.sc. thesis, Sharif University of Technology, Tehran, 2014.

[2] C. Bautista-Santiago, D. L. J.M. Díaz-Báñez, C. Peláez, and J. Urrutia. On covering a class with arbitrary disk. Technical report, 2008.

[3] S. Bereg, S. Cabello, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, and I. Ventura. The class cover problem with boxes. *Comput. Geom.*, 45(7):294–304, 2012.

[4] P. Berman and B. DasGupta. Complexities of efficient solutions of rectilinear polygon cover problems. *Algorithmica*, 17(4):331–356, 1997.

[5] P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness and satisfiability of bounded occurrence instances of SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, 10(022), 2003.

[6] P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, (049), 2003.

[7] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[8] A. Cannon and L. Cowen. Approximation algorithms for the class cover problem. *Ann. Math. Artif. Intell.*, 40(3-4):215–224, 2004.

[9] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 345–353, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[10] T. M. Chan and E. Grant. Exact algorithms and apx-hardness results for geometric packing and covering problems. *Comput. Geom.*, 47(2):112–124, 2014.

[11] T. M. Chan and N. Hu. Geometric red-blue set cover for unit squares and related problems. *Comput. Geom.*, 48(5):380–385, 2015.

[12] J. C. Culberson and R. A. Reckhow. Covering polygons is hard. *J. Algorithms*, 17(1):2–44, 1994.

[13] J. G. DeVinney. *The class cover problem and its application in pattern recognition.* PhD thesis, 2003.

[14] 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.

[15] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the vc-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, 2005.

[16] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[17] S. Har-Peled. Being fat and friendly is not enough. *CoRR*, abs/0908.2369, 2009.

[18] J. Håstad. Some optimal inapproximability results. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 1–10, 1997.

[19] D. S. Hochbaum and W. Maass. Fast approximation algorithms for a nonconvex covering problem. *J. Algorithms*, 8(3):305–323, 1987.

[20] D. Marchette. Class cover catch digraphs. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(2):171–177, 2010.

[21] W. Masek. On covering a class with arbitrary disk. Technical report, 1979.

[22] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[23] J. Nelson. A note on set cover inapproximability independent of universe size. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(105), 2007.

[24] J. Pach and G. Tardos. Tight lower bounds for the size of epsilon-nets. In *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 458–463, 2011.

Appendix A. Hardness of Approximation of MaxRM-3SAT

A.1. Reduction from MaxE3SAT-5 to MaxRM-3SAT

We use a reduction from MaxE3SAT-5 to show the inapproximability of MaxRM-3SAT. Given the fact MaxE3SAT-5 is APX-hard [16], we can observe that the following problem is also APX-hard: MaxE3SAT-5 even where every variable appears at least twice. Here we use this instance of MaxE3SAT-5.

Suppose that an instance ϕ of MaxE3SAT-5 is given with n variables and $m = 5n/3$ clauses. The following transformation F transforms ϕ to Φ, an instance of MaxRM-3SAT with M clauses and N variables.

F adds all the clauses of ϕ to Φ, then changes the name of variables and adds some clauses as described below.

First, F takes arbitrary orders on ϕ's variables and clauses. For each variable x, let $c_{x,i}$ be the ith clause that x appears in ϕ.

For each variable x and clause $c = c_{x,i}$, F replaces x with x_i or $\bar{x_i}$ in $c\prime$, the corresponding clause in Φ, as described in the following steps. In addition, let f be a mapping that shows if x replaced by x_i or $\bar{x_i}$; $f(x,i) = x_i$ or $\bar{x_i}$.

1. If $c = c_{x,i} = (x \vee y \vee z)$, F replaces x with $\bar{x_i}$ such that $c\prime = (\bar{x_i} \vee y \vee z)$ and $f(x,i) = \bar{x_i}$.
2. If $c = c_{x,i} = (x_i \vee t_2 \vee t_3)$, where either or both $t_2 = \bar{y}$ and $t_3 = \bar{z}$, F replaces x with x_i such that $c\prime = (x_i \vee t_2 \vee t_3)$ and $f(x,i) = x_i$;

After step 1 and step 2 are completed, F adds the clauses in steps 3 to Φ to have equality of x_i's instances with original x. For each variable x in ϕ, let n_x be the number of x_i's in Φ. If $n_x > 1$, then

3. F adds the following clauses to Φ for each $1 \leq i \leq n_x$: $(f(x,i) \vee \overline{f(x,t)})$, where $t = i + 1$ if $1 \leq i \leq n_x - 1$; $t = 1$ if $i = n_x$.

Now Φ is an instance of RM-3SAT. This is because it is a CNF formula, each clause of size 3 has one negated and one non-negated literal, each variable appears in one of the clauses of size 3, and each variable appears in exactly one of the clauses of size 2 in negated form, and exactly one of the clauses of size 2 in non-negated form. The number of clauses in Φ is M, $M = m + 3m = 4m$, where m is the number of clauses in ϕ. This is because the number of clauses of size 3 in Φ is m. Besides, the number of clauses of size 2 associated with each variable x is at most the number of times it appears in ϕ, so the total number of clauses of size 2 for all the variables is $3m = 5n$.

Lemma 8 *If there is an assignment for ϕ that satisfies $m-k$ clauses, then there is an assignment for Φ that satisfies at least $M - k$ clauses and can find such an assignment for Φ by having the assignment for ϕ.*

Proof. Consider the assignment for ϕ that satisfies at least $m - k$ clauses. Set the value of all the x_i's in Φ to the equivalent value of x in this assignment for ϕ by using f. e.g $x_i = x$ if $f(x,i) = x_i$, and $x_i = \bar{x}$ if $f(x,i) = \bar{x_i}$. Therefore, the only unsatisfied clauses of size 3 in Φ are the ones whose their corresponding clauses in ϕ are not satisfied. Besides, all the clauses of size 2 are satisfied in Φ. This is because all the values of x_i's are equivalent for any variable x, e.g. $f(x,i) = f(x,j)$ for any $1 \leq i,j \leq n_x$. This means either all the $f(x_i)$'s are true or all the $\overline{f(x,i)}$. Thus, at most, k clauses are unsatisfied by this assignment for Φ. □

Lemma 9 *If there is an assignment for Φ that leaves no more than k clauses unsatisfied, there is an assignment for ϕ that leaves no more than k clauses unsatisfied and can find such an assignment for ϕ by having the assignment for Φ.*

Proof. First, we explain how we can change any assignment for Φ with no more than k unsatisfied clauses to another assignment, in which for each variable x all the x_i's have equivalent value and there are no more than k unsatisfied clauses. Then, in the assignment for ϕ, we set the value of x to the value that all x_i's agreed on, and we show that this assignment does not leave more than k clauses unsatisfied.

For each x, we change the value of x_i's to the majority value of them. Now, we describe why this assignment for Φ has no more than k unsatisfied clauses. The number of unsatisfied clauses in Φ with this new assignment is, at most, the number of unsatisfied clauses of size 3 with the original assignment added to the number of satisfied clauses of size 3 in Φ that are no longer satisfied due to the change in x_i's values. For each x with equivalent values of x_i's in the original assignment, there is no change in the value of them in the new assignment and consequently no change by these variables in the number of satisfied clauses of size 3. For each x with non-equivalent values of x_i's in the original assignment, at most, two clauses of size 3 can be unsatisfied after changing their values to their equivalent majority value, as $i \leq 5$ and there are no more than two changes in x_i's values. On the other hand, if all the values of x_i's are equivalent, then the clauses of size 2 for each variable x are all true. But, if the values of x_i's are not equivalent, then, at least, two of their clauses of size 2 are unsatisfied. So, changing the value of x_i's to their majority satisfies, at least, two more clauses of size 2 for these variables. This means that changing the value of x_i's to their majority does not increase the number of unsatisfied clauses. Therefore, we have an assignment for Φ with no more than k unsatisfied clause and for each x all the x_i's are equivalent. Finally, for the assignment for ϕ, each variable x gets the majority equivalent value of x_i's in Φ, and there are no more than k unsatisfied clauses in this assignment for ϕ. □

A.2. Hardness of Approximation

Observation 2 *For any instance of MaxE3SAT-5 there is an assignment that satisfies at least half of the clauses.*

Proof of Theorem 2. Consider the two CNF formulas ϕ and Φ, an instance of MaxE3SAT-5 with m clauses and an instance of MaxRM-3SAT with M clauses, respectively. Assume the optimal solution for ϕ has $(m-k)$ true clauses. Lemma 8 implies that there is an assignment for Φ with at least $(M-k)$ true clauses. Assume that there is an algorithm that approximates MaxRM-3SAT with factor C_{RM-SAT}, where $C_{RM-SAT} \leq 1$. Then, the algorithm provides an assignment that has at least $C_{RM-SAT}(M-k)$ true

clauses and at most $M - C_{RM-SAT}(M-k)$ false clauses in Φ. By using Lemma 9, we can find an assignment that has at most $M - C_{RM-SAT}(M-k)$ false clauses and at least $m - M + C_{RM-SAT}(M-k)$ true clauses in ϕ. By knowing $M \leq 4m$, then the approximation ratio of this assignment in ϕ is at least $\rho = m - M + C_{RM-SAT}(M-k)/(m-k)$. By using Observation 2,$k \leq m$,

$$\rho \geq \frac{m - (1 - C_{RM-SAT})M - C_{RM-SAT}k}{m-k}$$
$$\geq \frac{m(4C_{RM-SAT} - 3) - C_{RM-SAT}k}{m-k}$$

If MaxE3SAT-5 is C_{3SAT5}-inapproximable [16], then we chose C_{RM-SAT} such that $\frac{m(4C_{RM-SAT}-3)-C_{RM-SAT}k}{m-k} \geq C_{3SAT5}$. Therefore, there is an algorithm to find C_{3SAT5}-approximation for MaxE3SAT-5, which is a contradiction.

Appendix B. Boxes Class Cover

Here we show that BCC is APX-hard in two ways: First, we demonstrate that the same presented reduction in [3] from Rectilinear Polygon Cover to BCC for showing the NP-hardness of this problem also shows that BCC is APX-hard. Second, we modify the reduction described in Section3.1 to show a reduction from MaxRM-3SAT to BCC.

Rectilinear Polygon Cover(RPC): Given a rectilinear polygon P, find a minimum size set of axis-aligned rectangles whose union is exactly P.

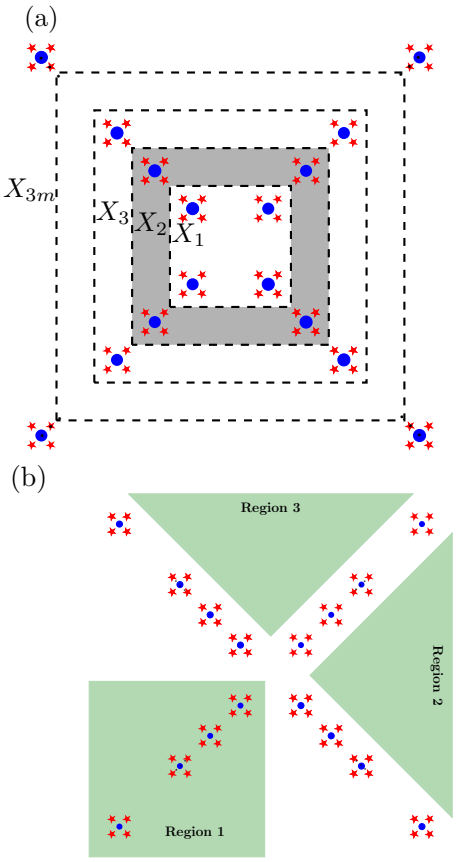
The idea of the reduction in [3] is to add blue points on the boundary and inside the polygon and add red points outside of the polygon in a way that the only possible non-empty rectangles are the ones that are inside P. Thus, any solution for BCC on these points gives a solution with the same size for RPC. Given that RPC is APX-hard [4], BCC is APX-hard.

B.1. Reduction from MaxRM-3SAT to BCC

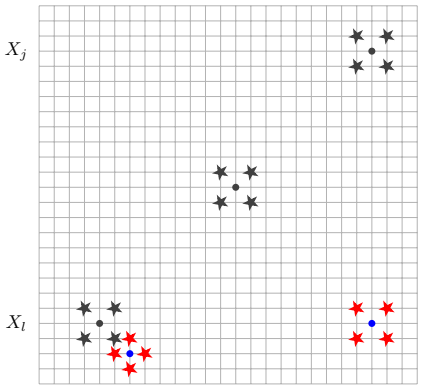
The idea of the reduction from MaxRM-3SAT to BCC is very similar to the idea of the reduction from MaxRM-3SAT to RBGSC[AARectangle]. For this reduction we keep all the same blue points, but we change the set of red points in a way that the set of all possible axis-aligned rectangles is limited to the *variable rectangles* and *clasue rectangles* and the rectangles in the optimal solution of BCC on this instance of the problem are the same as the rectangles in the optimal solution of RBGSC[AARectangle]. Figure 5 shows the location of these red points.

A rectangle t is a blue-rectangle if t contains only blue points but no red point. We can observe that blue-rectangles in any solution for BCC can be expanded to reach a red point without any change in the size of the solution. We call the set of all expanded possible axis-aligned rectangles *maximal rectangles*.

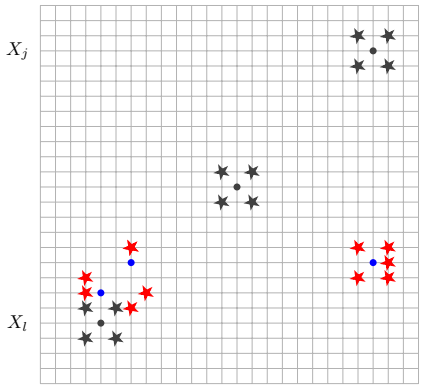
In this instance of BCC, the set of *maximal rectangles* that contain blue points is the same as the set of rectangles of RBGSC[AARectangle] in Section 3.1. In BCC, the goal is to minimize the number of these rectangles to cover all the blue points, which is the same as minimizing the number of covered red points in RBGSC[AARectangle] when each rectangle covers only one distinct red point. Therefore the same proof works here too.



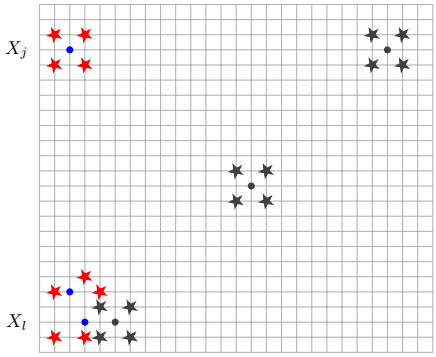
(c) In Region 1



(d) In Region 1



(e) In Region 1



(f) In Region 2

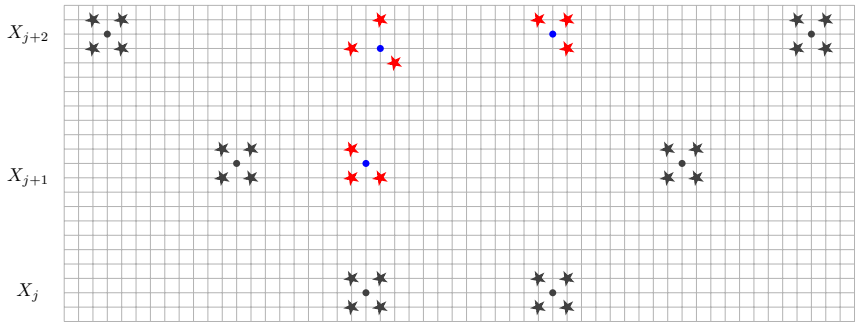


Figure 5: a) The points in the highlighted gray area are *variable points* for X_2. b) Highlighted green areas are divisions of the plane to Region 1-3. c) Added red points and blue points for $c = (X_j \vee \bar{X}_l)$, d) Added red points and blue points for $c = (X_j \vee X_l)$, e) $c = (\bar{X}_j \vee \bar{X}_l)$, f) Added red points and blue points for $c = (X_j \vee \bar{X}_{j+1} \vee \bar{X}_{j+2})$ (For $c = (\bar{X}_j \vee X_{j+1} \vee X_{j+2})$, added points are similar to part (f) but rotated by $-\pi/2$ in Region 3).

Relocating Units in Robot Swarms with Uniform Control Signals is PSPACE-Complete*

David Caballero Angel A. Cantu Timothy Gomez Austin Luchsinger Robert Schweller
Tim Wylie

Abstract

This paper investigates a restricted version of robot motion planning, in which particles on a board uniformly respond to global signals that cause them to move one unit distance in a particular direction on a 2D grid board with geometric obstacles. We show that the problem of deciding if a particular particle can be relocated to a specified location on the board is PSPACE-complete when only allowing 1×1 particles. This shows a separation between this problem, called the *relocation problem*, and the *occupancy problem* in which we ask whether a particular location can be occupied by any particle on the board, which is known to be in P with only 1×1 particles. We then consider both the occupancy and relocation problems for the case of extremely simple *rectangular* geometry, but slightly more complicated pieces consisting of 1×2 and 2×1 *domino* particles, and show that in both cases the problems are PSPACE-complete.

1 Introduction

The advanced development of microbots and nanobots has quickly become one of the most significant frontiers of our time. However, power and computation limitations at these scales often make autonomous robots infeasible and individually-controlled robots impractical. Thus, recent attention has focused on controlling large numbers of relatively simple robots. Many examples of large population robot swarms exist, ranging from naturally occuring magnetotactic bacteria [9, 11, 12] to manufactured light-driven "nanocars" [7, 13]. These particular microrobot swarms are manipulated uniformly through the use of external inputs such as light or a magnetic field. That is, all of the agents in the system react identically to the same global signal.

First proposed by Becker et al. [5], this model consists of movable polyominoes (as an abstraction of these nanorobots) that exist on a 2D grid board with "open" and "blocked" spaces. These polyominoes may be affected by global signals and step one unit distance when given a move command. Similar work has been shown in [4], where instead of moving one unit distance they travel maximally (referred to as "tilts"), which causes them to move linearly from one open location to another.

Previous Work. Before the tilt model was formally defined, there was research studying uniform control of particle swarms with precise movement [5]. Shortly after, investigation began on a version of particle swarm control where commands became limited and caused particles to move maximally [4]. In this work, the authors ask if any particle within a system can be moved to occupy a specified location. We refer to this problem as the *occupancy* problem. They prove that deciding the minimum number of moves needed to reconfigure one configuration of robots to another is PSPACE-complete. Recently in [2, 3], two additional natural questions for the model were proposed: the *relocation* and *reconfiguration* problems. The first asks whether a specified particle can be moved to a specified location. The second problem is to determine whether or not every particle in the system can be moved to its own specified location. In the later work the authors proved all of these problems to be PSPACE-complete even when limited to 1×1 tiles. These problems have also been investigated in the single-step model when considering limited directions. Recent work in [1, 6] shows that the relocation problem when limited to two or three directions and the reconfiguration problem when limited to two directions are both NP-complete. It was also shown that the *occupancy* problem is solvable in polynomial time in the single-step model even when all four directions are allowed.

Our Contributions. Our contributions are outlined in Table 1. We first show the *relocation problem* is PSPACE-complete with only 1×1 tiles by way of a reduction from a restricted version of the relocation problem within the *full-tilt* model, recently shown to be PSPACE-complete in SODA 2020 [2]. We then consider the case of domino shaped pieces, but with board geometry limited to being a single rectangle, and show that in this case both the *relocation* and *occupancy* problems are PSPACE-complete by a reduction from the problem of traversing a toggle-lock maze, shown to be PSPACE-complete in [8]. Videos of the constructions can be found at `https://asarg.hackresearch.com/main/cccg2020-Complexity`

*This research was supported in part by National Science Foundation Grant CCF-1817602.

Problem	Tile Size	Geometry	Result	Theorem
Occupancy	1×1	All	P	In [6]
Relocation	1×1	Connected	PSPACE-complete	Thm. 2
Occupancy/Relocation	1×1, 1×2	Rectangular	PSPACE-complete	Thms. 4,5

Table 1: An overview of the complexity results. For 1×1 polyominoes, the occupancy problem is in P, but the related problem of relocation is PSPACE-complete. We show that if 1×2 and 2×1 polyominoes (dominoes) are allowed, both of the problems are PSPACE-complete even with rectangular geometry.

2 Preliminaries

Board. A *board* (or *workspace*) is a rectangular region of the 2D square lattice in which specific locations are marked as *blocked*. Formally, an $m \times n$ board is a partition $B = (O, W)$ of $\{(x, y)|x \in \{1, 2, \ldots, m\}, y \in \{1, 2, \ldots, n\}\}$ where O denotes a set of *open* locations, and W denotes a set of *blocked* locations- referred to as "concrete." We classify the different board geometries according to the following hierarchy:

- Connected: A board where the set of open spaces O is a connected shape.
- Simple: A connected board is said to be *simple* if O has genus-0.
- Monotone: A simple board where O is either horizontally monotone or vertically monotone.
- Convex: A monotone board where O is both horizontally and vertically monotone.
- Rectangular: A convex board is *rectangular* if O is a rectangle.

Tile and Polyomino. A tile is a unit square centered on a non-blocked point on a given board. Formally a tile stores a coordinate on the board c and is said to occupy c. A *polyomino* is a finite set of tiles $P = \{t_1, \ldots t_k\}$ that is connected with respect to the coordinates occupied by the tiles in the polyomino. A polyomino that consists of a single tile is informally referred to as a "tile." In this work we only use single tiles and dominos which are polyominos consisting of two tiles.

Configurations. A configuration is an arrangement of polyominoes on a board such that there are no overlaps among polyominoes, or with blocked board spaces. Formally, a configuration $C = (B, P = \{P_1 \ldots P_k\})$ consists of a board B and a set of non-overlapping polyominoes P that each do not overlap with the blocked locations of board B.

Step. A *step* is a way to turn one configuration into another by way of a global signal that moves all tiles in a configuration one unit in a direction $d \in \{N, E, S, W\}$ when possible without causing an overlap with a blocked position, or another tile. Formally, for a configuration $C = (B, P)$, let P' be the maximal subset of P such that translation of all tiles in P' by 1 unit in the direction d induces no overlap with blocked squares or other tiles. A step in direction d is performed by executing the translation of all tiles in P' by 1 unit in that direction.

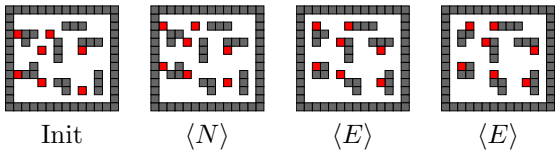


Figure 1: An example step sequence. The initial board configuration followed by the resulting configurations after an N step, E step, and then final E step.

We say that a configuration C can be *directly reconfigured* into configuration C' (denoted $C \rightarrow_1 C'$) if applying one step in some direction $d \in \{N, E, S, W\}$ to C results in C'. We define the relation $\rightarrow_*$ to be the transitive closure of $\rightarrow_1$ and say that C can be *reconfigured* into C' if and only if $C \rightarrow_* C'$, i.e., C may be reconfigured into C' by way of a sequence of step transformations. A related concept that is the focus of previous work is the *tilt* transformation in which a single direction d tilt consists of the repeated application of a direction d-step until the configuration is d-terminal. In this paper we focus on the step transition, but discuss connections to previous work using the tilt transformation.

Step Sequence. A *step sequence* is a series of steps which can be inferred from a series of directions $D = \langle d_1, d_2, \ldots, d_k \rangle$; each $d_i \in D$ implies a step in that direction. For simplicity, when discussing a step sequence, we just refer to the series of directions from which that sequence was derived. Given a starting configuration, a step sequence corresponds to a sequence of configurations based on the step transformation. An example step sequence $\langle N, E, E \rangle$ and the corresponding sequence of configurations can be seen in Fig. 1.

3 Hardness Results for Occupancy and Relocation

In this section we present our two PSPACE-completeness results. We first show the relocation problem is PSPACE-complete when allowing only 1×1 tiles by reducing from a restricted form of reloca-

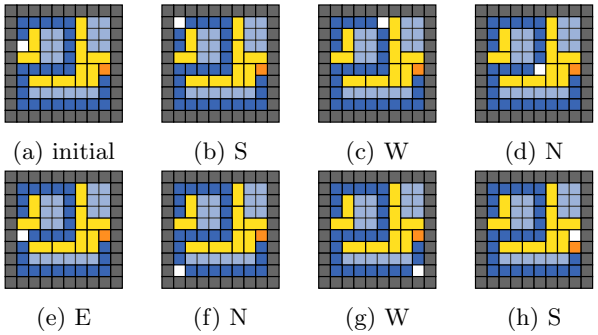


Figure 2: An example of an empty space moving through a configuration. The board geometry is just a rectangular frame. The dominoes along with many of the tiles (shown in lighter blue) are gridlocked and cannot move. We can see that through a sequence of tilts the space can move through the configuration and eventually allow the orange tile to change position.

tion within the full-tilt model, shown to be PSPACE-complete in [2]. Then, we show that both relocation and occupancy problems are PSPACE-complete when allowing 1×2 and 2×1 polyominoes even when restricted to a board with rectangular geometry. We show this by a reduction from the problem of moving a single robot through a *toggle-lock* maze [8]. Both of our PSPACE-hardness reductions utilize a common technique in which we consider an empty space in a mostly-full board as an agent. With this technique, isolated spaces now travel maximally across the board per step, similar to a single tile in the full-tilt model. This method is demonstrated in Figure 2.

3.1 Problem Definitions

Occupancy. The occupancy problem asks whether or not a given location can be occupied by any tile on the board. Formally, given a configuration $C = (B, P)$ and a coordinate $e \in B$, does there exist a step sequence such that $C \rightarrow_* C'$ where $C' = (B, P')$ and $\exists p \in P'$ that contains a tile that occupies coordinate e?

Relocation. The relocation problem asks whether a specified polyomino can be relocated to a particular position. Formally, given a configuration $C = (B, P)$, a polyomino $p \in P'$, and a coordinate $e \in B$, does there exist a step sequence such that $C \rightarrow_* C'$ where $C' = (B, P')$ and a tile in p occupies coordinate e?

3.2 Relocation with 1×1s

Recently, [2] proved that occupancy and relocation in the full-tilt model are PSPACE-complete with only 1×1 tiles. We can reduce directly from a modified version of the occupancy problem in full-tilt. The key idea in the reduction to invert the construction from [2] so that every space is replaced by a single tile, and vice versa. Now, the empty spaces act as the tiles in the original reductions and behave similarly to that in Figure 2 (although, this results in a board with no dominoes).

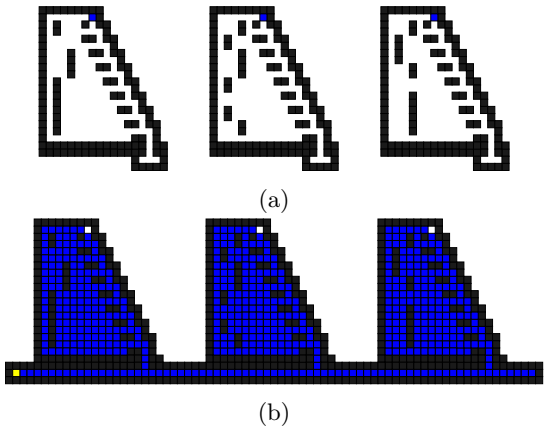


Figure 3: (a) An example input for the k-region relocation problem. (b) Reducing the k-region relocation problem to the relocation problem.

Lemma 1 *The relocation problem in the single-step model is in PSPACE.*

Proof. The problem can be solved by non-determinisitically selecting a movement from the available current movements until a tile is in the correct position. We only need to keep track of the current configuration between each move so the problem can be solved in NPSPACE which is known to equal PSPACE. □

Theorem 2 *The relocation problem in the single-step model is PSPACE-complete even when limited to only* 1×1 *tiles and connected geometry.*

Proof. To show hardness we reduce from a restricted version of the relocation problem under the *full-tilt* operation. The full-tilt model simply moves all pieces maximally in a given direction until colliding with a wall or other obstructed unit. In [2] the following restricted version of this problem, which we will call the *k-region relocation* problem, was shown to be PSPACE-complete [1] by way of a reduction from non-deterministic constraint logic [10]. In this problem we consider an input board configuration consisting of k disjoint regions, each with a single particle within each region. Further, we append a 1×3 enclosed region to the bottom row of each of these regions that includes a single central opening at

[1]This version of the problem was not explicitly formulated within the conference version of this paper, but this subproblem represents the key portion from which the hardness is derived. Key details and a formal proof is provided in Section 4.

the center leading the next higher row. See Figure 3a for an example. Given such an input, the *k-region relocation* problem asks if it is possible to move all k pieces into their corresponding 1×3 enclosed regions.

Given the PSPACE-hardness of *k-region relocation*, we now show the PSPACE-hardness of the relocation problem within our single-step model. The key idea is to apply the technique of filling each of the k disjoint regions with tiles, with the exception of the location of the given region's single particle. In this way, each step transition moves the empty particle in the same manner a full-tilt transition would maximally move a single particle (but in the opposite direction). Next, we connect the 1×3 output regions as shown in Figure 3b. In this way, the k empty spaces are able to reach the bottom-most row of the configuration if and only if the original *k-region relocation* input can relocate it's k pieces to the k output regions. With a final additional step the k spaces combine to create enough space for the target particle (shown in yellow) to move exactly k spaces to a designated relocation point. □

3.3 Complexity with Rectangular Board Geometry and Dominoes

In this section we relax the restriction on tile size and show both the occupancy and relocation problems are both PSPACE-complete even when restricted to rectangular board geometry, and with particles of size at most 2. We show this by reducing from a simple gadget model proposed in [8]. The authors show that the problem of relocating a single agent in a connected system of these gadgets is PSPACE-complete.

Gadget Basics. The gadgets used follow simple rules. They have two states, and contain tunnels that allow traversal through the gadgets. These tunnels exist in different types, such as the lock and toggle. A toggle tunnel can always be traversed in one direction, and on a state change that direction is reversed. The lock tunnel can be traversed in either direction when it is unlocked, and neither when it is locked. On a state change the lock tunnel will either lock or unlock. A gadget can contain multiple tunnels, each affected by the gadget's state changes. For our purpose we will use a crossing toggle lock, as shown in Figure 4a.

Crossing Toggle-Lock Domino Gadget. The Crossing Toggle-Lock Domino Gadget, shown in Figure 4b, enforces the same rules for traversal with two dominoes. When in the unlocked state the horizontal tunnel contains only 1×1 tiles and allows for the space to travel through it unblocked. When in the locked state there is a domino blocking the horizontal path. When a space attempts to pass through that path it is blocked by the domino and cannot continue through the gadget.

The vertical tunnel only allows traversal in one direction based on the state of the gadget. When in the unlocked state, traversal is allowed from south to north and if attempting to enter from the north, it is blocked by a domino. When in the locked state traversal is only allowed from north to south. Any complete traversal through the vertical tunnel will change the location of the dominoes in the tunnel and the state of the gadget.

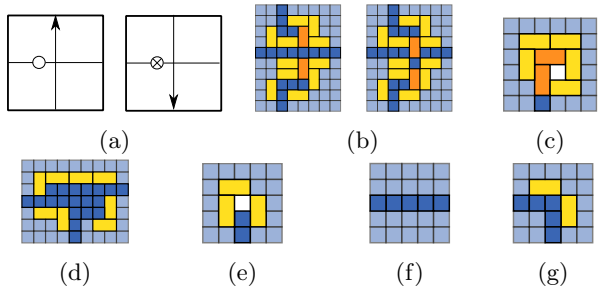


Figure 4: (a) Crossing Toggle-Lock (CTL) gadget in state 1 *left* and in state 2 *right*. The ⊗ represents the locked state of the lock tunnel. (b) The crossing toggle-lock gadget implemented in the single-step tilt model. The left image is in an open position, and the right is the closed position. (c) The goal gadget for occupation. The space can only be covered by a polyomino if another space is in the gadget. (The light blue tiles are used to fill up the board and keep polyominoes in the gadgets from moving. These tiles will never move) (d) The 3-way branching gadget that allows the space to enter at any of the 3 locations and exit at any other. (e) The Start Gadget. Intially contains the space that acts as the agent and has dominos to enforce that the space can only exit at one location. (f) The wire gadget, a group of tiles act as a medium for the space to travel through. (g) Corner Gadget used to allow the space to change directions.

Other Gadgets. In order to fully implement a CTL puzzle, we need a few other gadgets shown in Figures 4(d-g).

Branching Gadget. The other gadget required in the motion planning problem is a 3-way branching gadget. The gadget is shown in Figure 4d and connects all three locations and allows for movement between them. The way the gadget is set up is when entering from any point the space will be able to cycle around the edges of the gadget. At certain positions in the gadget the space will be able to exit out one of the locations.

Wire Gadget. The wire gadget shown in Figure 4f is just a group of single tiles. These tiles connect the other gadgets and allow the agent to travel through them.

Corner Gadget. The puzzle solvabilty problem allows for wires that turn. Since the agent travels the maximum distance possible before reaching a domino or the edge of the board we create a corner gadget (Figure 4g)

to allow the agent to stop and change direction.

Start Gadget. The start gadget (Figure 4e) is where the agent starts. When constructing the reduction the gadget contains the space that acts as the agent. The open position is surrounded on three sides by dominoes so the space can only exit from one side.

Goal Gadget. The goal gadget (Figure 4c) is the objective for the agent to reach. The gadget contains a second empty space that is surronded by dominoes. This space is the goal location. There is a horizontal domino that can be moved into this space if the agent reaches the goal gadget. The horizontal domino can only fill the goal location if the agent reaches the goal location.

Lemma 3 *The occupancy problem in single-step is PSPACE-hard with a rectangular board.*

Proof. Given an instance of a CTL puzzle we create a configuration by replacing each element of the CTL puzzle with one of our gadgets. We replace each CTL gadget with a crossing toggle-lock domino gadget and every 3-way intersection with a branching gadget. We also replace the start location with the start gadget and the goal location with the goal gadget. We finally connect these with wire gadgets and corner gadgets.

Our crossing toggle-lock domino gadget must behave the same as the CTL gadget. We can see that the space can only traverse the crossing toggle-lock domino gadget when an agent can traverse a CTL gadget in the same state. Observe that a space can travel through the horizontal tunnel when the gadget is in the unlocked state. While in this state observe that the space can only traverse the vertical tunnel from south to north since the north entrance is blocked by a domino in all directions. When traversing from south to north in this state we can see that the dominoes are able to move downward one step changing the state of the gadget to the locked state. Observe that in the locked state the horizontal tunnel is blocked by a vertical domino so a horizontal traversal in either direction is not possible. Also, observe that the space cannot traverse the vertical tunnel when entering from the south since it is blocked by a domino. When entering from the north in this state the space can traverse and changes the locations of the dominoes.

There exists a solution to the given instance of the CTL puzzle if and if only if there exists a solution to the occupancy problem on the given configuration. Since the crossing toggle-lock domino gadget has the same behavior of the CTL gadget, and the branching gadget allows a tile to enter and exit at any location, we can see that if the CTL puzzle is solvable then there exists a move sequence that solves the occupancy problem. Also since our gadgets behave the same as the gadgets in the CTL puzzle if there does not exist a solution to the CTL puzzle then there is no way for the space to reach the goal gadget and no way to solve the occupancy problem. □

Theorem 4 *The occupancy problem in single-step is PSPACE-complete when limited to rectangular board geometry if both* 1×1 *tiles and* 1×2 */* 2×1 *dominoes are included.*

Proof. We can see that the occupancy problem is in PSPACE in the same way as in Lemma 1 since we can non-deterministically select a valid move sequence. Through the reduction in Lemma 3 we show the problem is PSPACE-hard so the occupancy problem with the paramaters shown is PSPACE-complete. □

Corollary 5 *The relocation problem in single-step is PSPACE-complete even when limited to a rectangular board geometry when allowing* 1×1 *tiles and* 1×2 */* 2×1 *dominoes.*

Proof. We can see from Lemma 1 that the relocation problem is in PSPACE. The reduction from above can be extended to show the relocation problem is PSPACE-hard by asking if the horizontal domino in the goal gadget can reach the positon directly below it. □

4 Relocation Complexity in Full Tilt

This section is taken from [2] with the additional proof of *k-region relocation* hardness. To achieve this result we provide a polynomial time reduction from Non-Deterministic Constraint Logic [10]. We explain high level details of this construction along with key lemmas.

4.1 Non-Deterministic Constraint Logic

A constrant logic graph is a weighted directed graph with a constraint on each of the vertices [10]. The constraint specifies the minimum weight required from the edges directed in (the sum of the inflow) to any vertex. When given a graph, the usual problem studied is whether a particular edge can be "flipped"- the direction of the edge changed, i.e., is there a sequence of edge flips that maintain the constraints on all vertices, and allows the target edge to be flipped? This is a one-player unbounded game. The problem is still PSPACE-Complete when the edge weights are all strength 1 or 2, and vertices have max degree 3. We address the following equivalent problem.

Configuration-to-Configuration Problem. Given two states of a constraint graph G and G', does there exist a sequence of edge flips starting with G that results in G' [10].

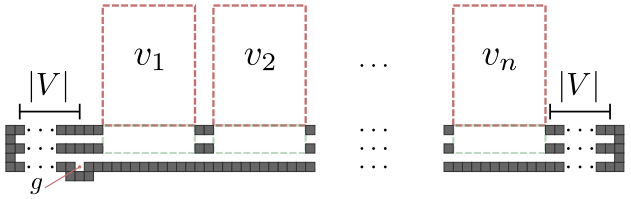


Figure 5: An overview of the layout of the different components for the reduction. The dotted red lines represent where each of the vertex gadgets go (not to scale), the dotted green boxes below denote the geometry specific to each vertex to force the state tile into the top row (the vertex was in the wrong state) unless the vertex is in the state specified by the target configuration. The bottom row requires all $|V|$ state tiles in order for a tile to get into the goal location g.

4.1.1 Vertex Gadget

Assuming a max degree of three, there are 8 possible arrangements of in/out edges. Define the vertex *state* as a label from 0 to 7 determined by the directions of its incident edges. A vertex gadget contains a single 1×1 tile refered to as the *state tile*, a *transition area*, and a number of *state gadgets* equal to the number of legal states of that vertex. Since there are eight states, there are eight basic paths in the gadget that the state tile could be in representing the vertex's state.

Flipping an edge is represented by a move sequence performed while in a valid state that moves the state tile from one state path to another, which happens simultaneously in two vertex gadgets since an edge connects two vertices. This edge flip happens in all vertex gadgets, but if the edge is not incident to that vertex, there is no effect on the path of the state tile.

4.1.2 Goal Area

An overview of the reduction layout is in Figure 5 where the goal area is shown at the bottom of all the vertex gadgets. Once all the tiles are in positions that represent the target configuration, the tiles can be extracted into the goal area though the bottom of a state gadget. After extraction the tiles enter the goal area. The goal area consists of two rows. The valid row and the invalid row. The invalid row (top row) traps any tiles that enter when a vertex was not in the specified (in the target configuration) state. If there exists a solution to the Configuration-to-Configuration Problem then all tiles will be able to reach the valid row.

Lemma 6 *After performing a move sequence to flip an edge, only the two vertex gadgets representing vertices incident to that edge will have their state tile change state paths. All other vertex gadgets will have their state tile stay in the same state path.*

Lemma 7 *If a vertex enters an illegal state, the representative vertex gadget's state tile will be trapped in an 'illegal' state path and cannot be extracted.*

4.2 Hardness of k-region Relocation

In this section we will describe how to modify the reduction from [2] to show hardness for the *k-region relocation problem*.

k-region relocation. The k-region relocation problem asks: given a board with k disjoint regions each containing a single tile, and a set of positions in each region called goal areas, does there exist a move sequence that relocates all tiles to their goal area?

Theorem 8 *The k-region relocation problem in the full-tilt model is PSPACE-hard.*

Proof. First, note in the orginal reduction that the goal location may be filled if and only if each tile is extracted from its vertex gadget and enters the goal row. This means that the problem of "Can each tile be extracted from it's vertex gadget?" is PSPACE-hard. Now consider the board used for the proof of hardness for the occupancy problem in [2]. Each vertex gadget is only connected to the others through the two rows at the bottom of the construction. Both of these rows can be removed and replaced with the 1×3 regions described in Theorem 2. The k-many 1×3 rows (which replaced the goal row) can now be reached if and only if each tile can be extracted from it's goal gadget. □

5 Future Work

There are a number of directions for future work. We show that with only 1×1 tiles the *relocation* problem is PSPACE-complete with a connected board. Relocation and occupancy become PSPACE-complete when restricted to a rectangular board but allowing for larger pieces. How much power do these constraints remove? Do these problems become easier when only restricting either the board geometry or the number of larger pieces (i.e., constant number of dominoes), or are they still hard?

References

[1] Jose Balanza-Martinez, David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie, *Relocation with uniform external control in limited directions*, The 22^{nd} Japan Conference on Discrete and Computational Geometry, Graphs, and Games, JCDCGGG, 2019, pp. 39–40.

[2] Jose Balanza-Martinez, Timothy Gomez, David Caballero, Austin Luchsinger, Angel A. Cantu, Rene Reyes, Mauricio Flores, Robert T. Schweller, and Tim Wylie, *Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces*, Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA'20, SIAM, 2020, pp. 2625–2641.

[3] Jose Balanza-Martinez, Austin Luchsinger, David Caballero, Rene Reyes, Angel A. Cantu, Robert Schweller, Luis Angel Garcia, and Tim Wylie, *Full tilt: Universal constructors for general shapes with uniform external forces*, Proceedings of the 30^{th} Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'19, 2019, pp. 2689–2708.

[4] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Golnaz Habibi, and James McLurkin, *Reconfiguring massive particle swarms with limited, global control*, Algorithms for Sensor Systems (Berlin, Heidelberg) (Paola Flocchini, Jie Gao, Evangelos Kranakis, and Friedhelm Meyer auf der Heide, eds.), Springer Berlin Heidelberg, 2014, pp. 51–66.

[5] Aaron T. Becker, Golnaz Habibi, Justin Werfel, Michael Rubenstein, and James McLurkin, *Massive uniform manipulation: Controlling large populations of simple robots with a common input signal*, 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nov 2013, pp. 520–527.

[6] David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie, *Hardness of reconfiguring robot swarms with uniform external control in limited directions*, ArXiv e-prints (2020), arxiv:2003.13097.

[7] Pinn-Tsong Chiang, Johannes Mielke, Jazmin Godoy, Jason M. Guerrero, Lawrence B. Alemany, Carlos J. Villagómez, Alex Saywell, Leonhard Grill, and James M. Tour, *Toward a light-driven motorized nanocar: Synthesis and initial imaging of single molecules*, ACS Nano **6** (2012), no. 1, 592–597, PMID: 22129498.

[8] Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy, *Computational complexity of motion planning of a robot through simple gadgets*, 9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy, 2018, pp. 18:1–18:21.

[9] Ouajdi Felfoul, Mahmood Mohammadi, Louis Gaboury, and Sylvain Martel, *Tumor targeting by computer controlled guidance of magnetotactic bacteria acting like autonomous microrobots*, 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sep. 2011, pp. 1304–1308.

[10] Robert A. Hearn and Erik D. Demaine, *The nondeterministic constraint logic model of computation: Reductions and applications*, Proceedings of the 29th International Colloquium on Automata, Languages and Programming (London, UK, UK), ICALP '02, Springer-Verlag, 2002, pp. 401–413.

[11] Sylvain Martel, *Bacterial microsystems and microrobots*, Biomedical Microdevices, vol. 14, 2012, pp. 1033–1045.

[12] Sylvain Martel, Samira Taherkhani, Maryam Tabrizian, Mahmood Mohammadi, Dominic de Lanauze, and Ouajdi Felfoul, *Computer 3d controlled bacterial transports and aggregations of microbial adhered nano-components*, Journal of Micro-Bio Robotics **9** (2014), no. 1, 23–28.

[13] Yasuhiro Shirai, Andrew J. Osgood, Yuming Zhao, Kevin F. Kelly, and James M. Tour, *Directional control in thermally driven single-molecule nanocars*, Nano Letters **5** (2005), no. 11, 2330–2334, PMID: 16277478.

Building Patterned Shapes in Robot Swarms with Uniform Control Signals*

David Caballero Angel A. Cantu Timothy Gomez Austin Luchsinger Robert Schweller
Tim Wylie

Abstract

This paper investigates a restricted version of robot motion planning, in which particles on a board uniformly respond to global signals that cause them to move one unit distance in a particular direction. We look at the problem of assembling patterns within this model. We first derive upper and lower bounds on the worst-case number of steps needed to reconfigure a general purpose board into a target pattern. We then show that the construction of k-colored patterns of size-n requires $\Omega(n \log k)$ steps in general, and $\Omega(n \log k + \sqrt{k})$ steps if the constructed shape must always be placed in a designated output location. We then design algorithms to approach these lower bounds: We show how to construct k-colored $1 \times n$ lines in $O(n \log k + k)$ steps with unique output locations. For general colored shapes within a $w \times h$ bounding box, we achieve $O(wh \log k + hk)$ steps.

1 Introduction

In this paper we investigate a model of robot motion planning first proposed by Becker et. al. [7] in which n robots exist on a 2D grid consisting of "open" and "blocked" spaces, and are controlled by way of uniform control signals which tell all robots to move one step in any one of the four cardinal directions. This model, which we call the single-step model, has important applications for the scalable development of microbot and nanobot swarms due to the simplified method of control [9,11]. While previous work in this model has investigated how to build general shapes [7], and the hardness of relocation related problems [1], here we focus on the problem of quickly rearranging the robots into a desired colored pattern (with an arbitrary shape).

In particular, our problem is as follows. Given a color palette of k distinct colors, as well as a bounding box of width w and height h, our goal is to design a *universal* board configuration (a board with open and blocked locations, as well as specified locations for a set of robots each assigned one of the k colors) with the property that any pattern fitting within a $w \times h$ bounding box can be assembled (the robots can be reconfigured into the provided pattern) in a near-optimal number of steps.

*This research was supported in part by National Science Foundation Grant CCF-1817602.

Our results. We first focus on a special case class of patterns consisting of $1 \times n$ lines over k colors. We provide a board that can assemble any $1 \times n$ patterned line over k colors within $O(n \log k + k)$ steps, along with showing a lower-bound of $\Omega(n \log k + \sqrt{k})$ under the assumption that the board must always place the output pattern in the same location. We extend this to general 2D shapes of size n and provide a construction achieving $O(wh \log k + hk)$ steps, which for dense shapes of size n is comparable to the lower bound of $\Omega(n \log k + \sqrt{k})$.

Previous Work. The single-step model of this paper was first studied in [7] where it was shown how to reconfigure n robots into any size n shape within $O(n^2)$ steps given a single blocked location. An additional line of related research considers global movement signals, but requires that all pieces move maximally in the input direction [4]. This line of research has explored building shapes [2,3,6,8,10], performing computation [5], as well complexities for reconfiguration and relocation of particles [2–5]. Additionally, [12] considers the reconfiguration of rectangular patterns of n colored robots within $O(n^2)$ steps. While closely related to our work, this work differs from the problem we are considering in that 1) they consider the maximal-movement of particles, and 2) we are attempting to build arbitrary patterns, while they are rearranging a given set of pieces (meaning the number of each color in the pattern is fixed). We also consider general shaped patterns and striving for near-optimal construction times, and are not attempting to reconfigure all pieces on the board.

2 Preliminaries

Board. A *board* (or *workspace*) is a rectangular region of the 2D square lattice in which specific locations are marked as *blocked*. Formally, an $m \times n$ board is a partition $B = (O, W)$ of $\{(x, y) | x \in \{1, 2, \ldots, m\}, y \in \{1, 2, \ldots, n\}\}$ where O denotes a set of *open* locations, and W denotes a set of *blocked* locations- referred to as "concrete."

Tiles/Robots. A tile/robot is a labeled unit square centered on a non-blocked point on a given board. Formally, a tile is an ordered pair (c, a) where c is a coordinate on the board, and a is a label.

Configurations. A configuration is an arrangement of tiles on a board such that no tiles occupy the same

Result	Step Complexity		Theorem
	Lower	Upper	
Patterned Lines	$\Omega(n \log k + \sqrt{k})$	$\mathcal{O}(n \log k + k)$	Thms. 3, 4
General Patterns	$\Omega(n \log k + \sqrt{k})$	$\mathcal{O}(wh \log k + hk)$	Thms. 3, 5

Table 1: Construction Results. The patterned lines result is for $1 \times n$ lines using k colors. The general patterns result is for k-colored size-n $w \times h$-bounded shapes.

location, or occupy blocked board spaces. Formally, a configuration $C = (B, P)$ consists of a board B and a set of tiles P whose coordinates do not overlap each other, or with blocked locations of board B.

Step. A *step* is a way to turn one configuration into another by way of a global signal that moves all tiles in a configuration one unit in a direction $d \in \{N, E, S, W\}$ when possible without causing an overlap with a blocked position, or another tile. Formally, for a configuration $C = (B, P)$, let P' be the maximal subset of P such that translation of all tiles in P' by 1 unit in the direction d induces no overlap with blocked squares or other tiles. A step in direction d is performed by executing the translation of all tiles in P' by 1 unit in that direction.

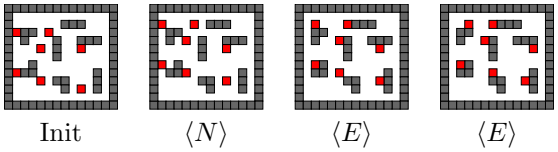
We say that a configuration C can be *directly reconfigured* into configuration C' (denoted $C \rightarrow_1 C'$) if applying one step in some direction $d \in \{N, E, S, W\}$ to C results in C'. We define the relation $\rightarrow_*$ to be the transitive closure of $\rightarrow_1$ and say that C can be *reconfigured* into C' if and only if $C \rightarrow_* C'$, i.e., C may be reconfigured into C' by way of a sequence of step transformations. A related concept that is the focus of previous work is the *tilt* transformation in which a single direction d tilt consists of the repeated application of a direction d-step until the configuration is d-terminal. In this paper, we focus on the step transition, but discuss connections to work using the tilt transformation.

Step Sequence. A *step sequence* is a series of steps which can be inferred from a series of directions $D = \langle d_1, d_2, \ldots, d_k \rangle$; each $d_i \in D$ implies a step in that direction. For simplicity, when discussing a step sequence, we just refer to the series of directions from which that sequence was derived. Given a starting configuration, a step sequence corresponds to a sequence of configurations based on the step transformation. An example step sequence $\langle N, E, E \rangle$ and the corresponding sequence of configurations can be seen in Figure 1a.

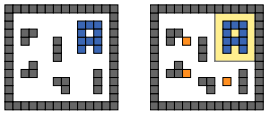
Universal Configuration. A configuration C' is universal to a set of configurations $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ if and only if $C' \rightarrow_* C_i, \ \forall \ C_i \in \mathcal{C}$.

Shape/Pattern. We define a shape to be a connected subset $S \subset \mathbb{Z}^2$. We define a pattern to be a tuple (S, L), where S is a shape and $L : S \rightarrow A$ is a total function that maps each point to a label in a set of labels A.

Configuration Representation. A configuration may be interpreted as having constructed a "shape" in a natural way. A configuration C *strongly* represents shape S if the collection of all tile coordinates in C is exactly the set of points of some translation $t(S)$.



(a) An example step sequence



(b) Representation Examples

Figure 1: (a) An example step sequence. The initial board configuration followed by the resulting configurations after an N step, E step, and then final E step. (b) Configuration Representation Examples. Both of these configurations are different representations of the shape "A." First, we show a strong representation where every tile in the configuration contributes to the shape. Then we show a regional representation. The yellow square represents the output region, and the orange tiles represent additional polyominoes on the board which do not count towards shape representation (as they are not in the output region).

An alternate form of representation allows for a rectangular region of the board to be deemed the *output* region. Here, we say a configuration *regionally* represents a shape S *w.r.t.* output region T if the collection of all tile coordinates in T is exactly the set of points of some translation $t(S)$. Figure 1b illustrates the different types of representations. In the regional representation, any tiles outside of the output region are ignored.

We extend this idea of shape representation to include patterns. A configuration C represents a pattern (S, L) if C represents S and there exists a translation t, such that for all tiles $(c, a), L(t(c)) = a$. The idea of regional representation of a pattern extends in the same way.

Universal Pattern Builder. Given the concept of pattern representation, a configuration C' is *universal* for a set of patterns $\mathcal{S}$ if and only if there exists a set of configurations $\mathcal{C}$ such that 1) each $S \in \mathcal{S}$ is represented

by a unique $C \in \mathcal{C}$ and 2) C' is universal for $\mathcal{C}$.

We say that C' is *regionally universal* for $\mathcal{S}$ *w.r.t.* output region T if $\forall S \in \mathcal{S}\ \exists\ C \in \mathcal{C}$ s.t. C regionally represents S *w.r.t.* T. Further, C' achieves *unique placement* if output region T is exactly the size of the minimum bounding box which can contain any of the patterns in $\mathcal{S}$. In this paper we focus on designing regionally universal configurations for general patterns fitting within a $w \times h$ bounding box.

Worst-Case Step Complexity for Universal Configurations. Given a universal configuration C, the worst-case step complexity is the maximum number of steps required to reconfigure C into some element from its universe set. Consider a universal configuration C over a set of configurations U. For each $u \in U$, let $d(C, u)$ denote the length of the smallest step-sequence from C to u. The worst-case step complexity of C over U is defined to be $\max\{d(C, u) | u \in U\}$.

3 Fast Universal Constructors: Patterns

We now focus on building shapes with a desired color pattern. To model this, we specify each robot in the system to have a designated color from a given set of k colors. Our goal is then to design configurations that allow quick reconfiguration of the robots into a specified shape with a specified color pattern. We start with an analysis of some lower bounds for any k-color pattern constructors in Section 3.1. We then derive upper bounds for linear patterns in Section 3.2, general patterns in Section 3.3. Accompanying videos for these constructions can be found at `https://asarg.hackresearch.com/main/CCCG2020-Patterns`.

3.1 Lower Bounds on Patterns

Lemma 1 *For a given set of n distinct points from the 2D integer lattice, consider the corresponding set of all size-n colored patterns over those points using at most k distinct colors. Any universal configuration for such a set of patterns has worst-case step complexity $\Omega(n \log k)$.*

Proof. There are k^n distinct k-color patterns over n points. Therefore, any universal configuration for this set of patterns must be universal to a set of k^n configurations. The maximum number of distinct configurations reachable from an initial configuration C' within r steps is upper bounded by

$$\sum_{i=0}^{r} (4^i) = \frac{4^{r+1} - 1}{3}.$$

Thus C' must satisfy that $\frac{4^{r+1}-1}{3} \geq k^n$, implying that $r = \Omega(n \log k)$. □

Lemma 2 *Any universal configuration for all k-colored patterns over a size-n shape with unique placement has worst-case step complexity $\Omega(\sqrt{k})$.*

Proof. Consider a unique placement universal constructor for all k-colored patterns over some size-n shape. As this is a unique placement constructor, the output zone is a fixed region of size exactly the bounding box of the size-n shape. Select an arbitrary point $p = (x, y)$ within the output region that is covered by the size-n shape when inscribed within the output region. Let $d = \lfloor \frac{\sqrt{k}}{4} - 1 \rfloor$ and note that the number of points within (Manhattan) distance d of (x, y) is strictly less than k. Therefore, there must be one color c for which all tiles of color c are at least distance d from point (x, y). Further, as this system is universal for all k-colored patterns over the target shape, and the unique placement restriction enforces the output shape into a fixed position for each represented pattern, there exists a pattern in the universe for which the color c must be placed at position (x, y). The step-sequence to place a color c tile at location (x, y) requires at least $d = \Omega(\sqrt{k})$ steps, and therefore requires at least $\Omega(\sqrt{k})$ steps to finish this pattern. □

Theorem 3 *Any universal configuration for all k-colored patterns over a shape of size-n has worst-case run-time at least $\Omega(n \log k)$. If the configuration satisfies the unique placement requirement, the worst-case run-time is at least $\Omega(n \log k + \sqrt{k})$.*

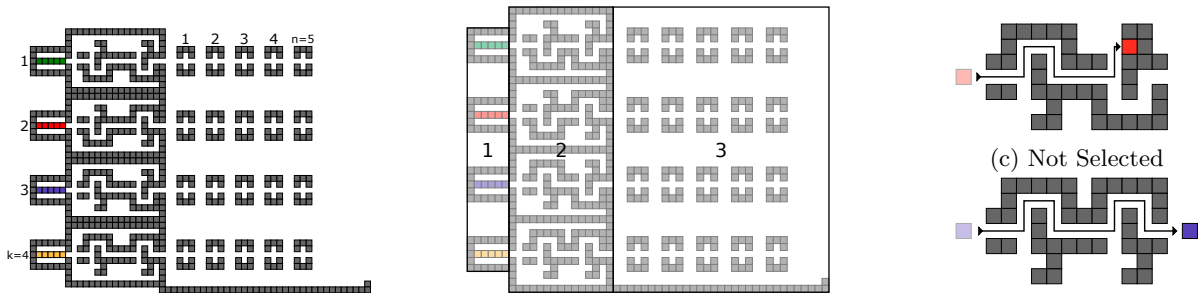
Proof. This follows from Lemma 1 and Lemma 2. □

3.2 Fast Linear Patterns

For our first positive result on universal pattern building we focus on the case of linear $1 \times n$ shapes over k colors. We construct a universal configuration with worst-case run time of $\mathcal{O}(n \log k + k)$ (Theorem 4), which is reasonably close to the lower bound of $\Omega(n \log k + \sqrt{k})$ shown in Theorem 3, and optimal in the case where $n \geq k$. The linear pattern constructor is made up of three sections: *fuel chambers*, *bit selectors*, and *holding chambers*.

Fuel Chambers. This section of the constructor consists of the fuel chambers, where each are $3 \times n$ open spaces surrounded by concrete with an opening on the center right. Morever, each chamber contains a $1 \times n$ line of robots of one color. Using the opening on the right side of each chamber we can "chop" off one robot at a time. By chopping off a robot from each chamber in parallel, we transmit a column of k differently colored robots into section 2.

Bit Selectors. The *bit selectors* are gadgets used to assign a unique *bit-string* to each colored robot entering the section. These bit-strings are created by the unique combination of two smaller gadgets called the



(a) Constructor (b) Sections (c) Not Selected (d) Selected

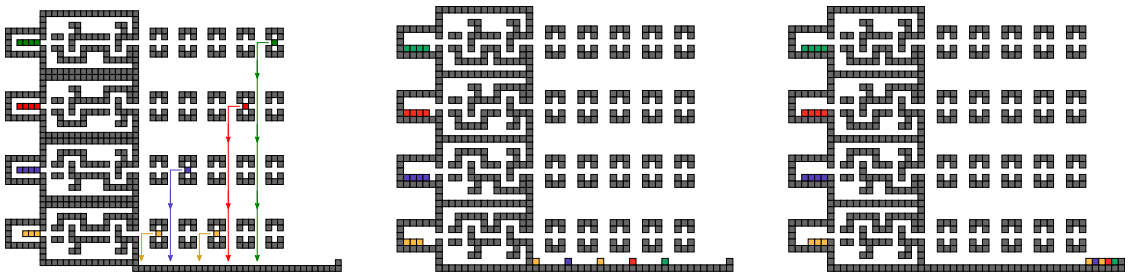
Figure 2: (a) The linear patterns constructor and (b) the different sections. Section 1 consists of the fuel chambers. Section 2 consists of bit-selector gadgets. Section 3 consists of tile holding chambers, as well as a concrete floor where the line will be assembled. (c) Example bit selection where one robot is extracted via execution of its unique bit-string.

up-select and *down-select*. Each of these smaller gadgets has an open space path from one side to the other such that each of their paths are the opposite of the other smaller gadget. The incorrect path causes a robot to get stuck. This idea is demonstrated in Figure 2d, where one robot can sucessfully traverse the bit selectors at the cost of the other robot stopping in the up-select gadget. Therefore, for all k robots entering this section from the fuel chamber, we can design a unique combination of up-select and down-select gadgets such that traversal of any robot through their respected gadgets will yield that robot on the other side, while all others stay within their gadgets. Therefore, bit selectors consisting of $\log k$ bits each are needed to yield a unique bit-string per robot, each of which creates an open space path from one side to the other of length $\mathcal{O}(\log k)$.

Holding Chambers. Each individual robot enters section 3 through the left side at possibly different heights since each colored robot comes from a different bit selector. There are nk holding chambers in the output area of the bit selectors, where each holding chamber is a 1×3 open space surrounded by concrete tiles, with an open space path from the center left to right. These chambers hold the robots in place while another robot is being extracted from the fuel chambers. After outputting a robot from a bit selector gadget, we place the robot in the closest holding chamber to the right. After placing the new robot in a holding chamber, we address the unselected robots that were blocked in the bit selectors. The unused robots are placed back into the fuel chambers by the sequence $\langle W^4, N^2, W^{\mathcal{O}(\log k)}, S^8, W^{\mathcal{O}(\log k)}, N^4, W^2, N, E^{\mathcal{O}(\log k)}\rangle$. After returning the unselected robots to their fuel chambers, we continue the building process. The sequence to extract robots and traverse the robot through the bit selector gadgets also moves all robots in a holding chamber to the next holding chamber on their right. After the n^{th} robot has been placed in section 3, we combine them by extracting them from the holding chambers and placing them all on the concrete floor. Then, we push them together using the single concrete tile on the right of this floor. Figure 3 shows an example of this sequence.

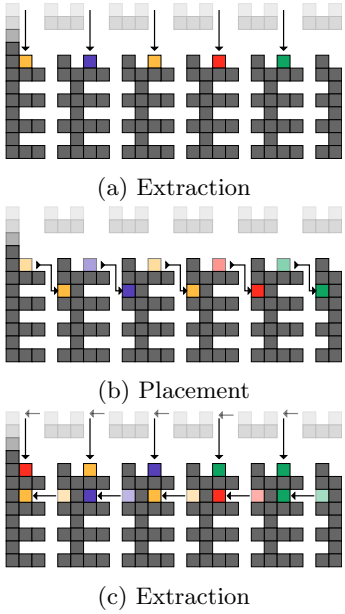
Theorem 4 *For any positive integers n and k, there exists a regionally universal configuration for all $1 \times n$ k-colored lines with worst-case step complexity $\mathcal{O}(n \log k + k)$. Moreover, this configuration obtains unique placement and has board-size $\mathcal{O}(n + \log k) \times \mathcal{O}(k)$.*

Proof. Above we describe a configuration $C = (B, P)$ such that it consists of three sections. The first section pertains to the fuel chambers, which is used to hold k $1 \times n$ lines of robots, one line for each color, in separate chambers. It follows that a single robot can be extracted from each of these chambers, resulting in a column of k robots entering the third section. The third section consists of the bit-string gadgets, where each receives one of the k robots. We have shown that the bit-string gadgets each have a specific unique sequence that takes the robot from the left side to the right side such that performing the sequence of one bit-string gadget will make all robots, save for the one that is within that bit-string gadget, stuck in one of the compartments in the bit selectors of the other bit-string gadgets. Therefore, it is possible to send one robot to the third section in $\mathcal{O}(\log k)$ steps. The holding chambers in the third section are used to hold the robots in place while the next robots are being extracted from the former two sections. Together, these sections can place n robots in the holding chambers in the third section, after which we can remove these robots from the holding chambers and combine them to form a line at the bottom side of the third section. Placing the robots at the bottom of the third section takes $\mathcal{O}(k)$ steps, while combining them takes $\mathcal{O}(n)$ steps. Therefore, the configuration C = (B, P) is a universal configuration for all $1 \times n$ k-colored lines



(a) Holding Chambers (b) Extraction (c) Line Building

Figure 3: Line building depicted.



(a) Extraction

(b) Placement

(c) Extraction

Figure 4: Line holders depicted. After each line is built and extracted from the holding chambers, we place them in the first row of the line holder. This, in parallel, will move each line already within the line holder down into the next row.

with worst-case runtime $\mathcal{O}(n \log k + k)$. Moreover, this configuration has board size $\mathcal{O}(n + \log k) \times \mathcal{O}(k)$ and achieves unique placement w.r.t. a $1 \times n$ rectangular output region located at the bottom-right of the board, along the concrete floor. □

3.3 General Patterns

We now generalize our line pattern construction to general shapes over k colors. For given positive integers n, h and w we focus on size-n shapes fitting in a $h \times w$ bounding box.

Line Holders. For general shapes, we replace the south concrete floor of section 3 of the line pattern builder with the *line holder* depicted in Figure 4. As shown, each new line built can be moved into the line holder. If some lines are inside the line holder already, those lines will move in parallel to the next chamber below whenever a new line is added to the line holder. After all lines have been built, we extract them, yielding essentially a general $w \times h$ pattern, but with a constant vertical and horizontal gap between tiles. Further, by adding in an "empty" color chamber, we can include empty spaces within this pattern, yielding a general patterned shape. Finally, to remove the gaps in the shape, we apply a *funneling* operation, described in Section 4.

Theorem 5 *For positive integers w and h, each greater than some constant, and positive integer k, there exists a regionally universal configuration for any k-colored size-n shape fitting within a $h \times w$ bounding box with worst-case step complexity $\mathcal{O}(wh \log k + hk)$ and board size $\mathcal{O}(wh + \log k) \times \mathcal{O}(\max(h, k))$.*

Proof. The k-colored shape constructor is a simple extension from the $1 \times n$ constructor. The main addition of the line constructor is the line holders at the bottom of the third section. Each different line we construct can be held inside one of these different line holders in order to build another line. After each line is made, we can move that line into the line holders and at the same time move any line already in the line holders down one row. After each line is built, we can extract them and send them through the funneling gadget in order to remove the *constant* amount of space between each tile. With the inclusion of "empty" tiles, we obtain general patterned shapes. The details of the funneling gadget are presented in 4. Therefore, the configuration $C = (B, P)$ is a universal configuration for all k-colored size-n shapes fitting within a $h \times w$ bounding box with worst-case step complexity $\mathcal{O}(wh \log k + hk)$. Moreover, this configuration has board size $\mathcal{O}(n + \log k) \times \mathcal{O}(k)$ and achieves unique placement w.r.t. a $w \times h$ output region located just above the funneling gadget. □

4 Funneling Gadget

The *funneling* gadget is designed to take a group of robots separated by a constant amount of spaces and co-

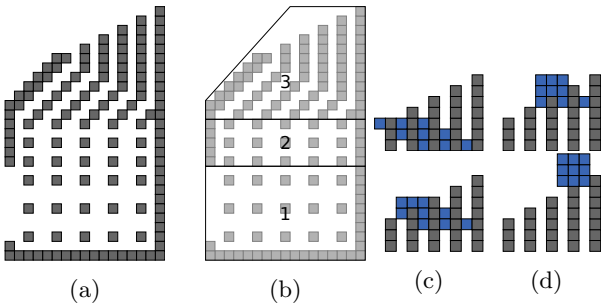


Figure 5: (a) An example funneling gadget for a 3×3 shape. (b) Basic functional sections of the funneling gadget. (c-d) Repeating the sequence $\langle N, E \rangle$ will yield the shape on the outside of the funneling gadget.

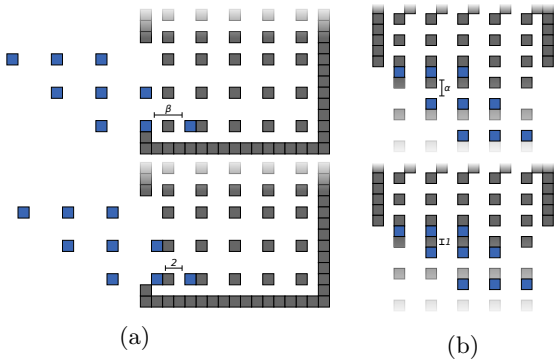


Figure 6: (a-b) Reducing the horizontal distance between the rightmost column of robots. (c-d) Reducing the vertical distance between the topmost row of robots.

alesce them into a desired shape. The architecture and sections of the funneling gadget are illustrated in Figure 5. Let α and β be constants equaling the largest number of vertical and horizontal spaces, respectively, that separates a robot from its neighbor in the shape. The first section of the funneling gadget reduces β so that the largest horizontal separation between two robots is two. Section two takes the group of robots from the former section and reduces α until it is one. The third section finally reduces α and β to zero, and outputs the group of robots as the desired shape outside the funneling gadget. However, this process skews the shape in one direction. This effect can be countered if the input group of robots are instead skewed in the *opposite* direction before passing them through the funneling gadget.

Section One. Section one consists of a grid-like organization of concrete tiles that are themselves spaced out vertically by α but horizontally by two, as shown in Figure 6. To reduce β to two, we place the rightmost column of the group of robots between the two leftmost columns of concrete tiles (Figure 6a). By stepping in the $\langle E \rangle$ direction enough times, the second-to-rightmost column of the group of robots will meet the leftmost

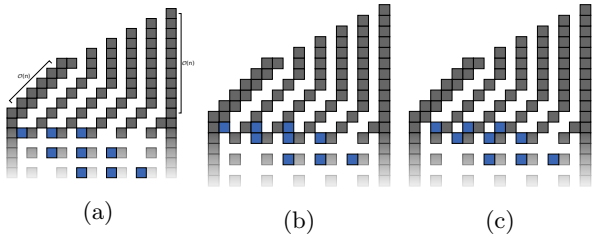


Figure 7: (a-b) Making the rows of robots adjacent by using section three.

column of section one, reducing the spaces between the two columns of robots. After repeating this process for every column of robots, section one will contain within itself the group of robots vertically separated by α and horizontally separated by a distance of two.

Section Two. Section two is a grid-like configuration of concrete tiles that are vertically separated by one space and horizontally separated by two spaces. The same basic process is applied here, but we instead place the rows of robots in between the rows of concrete tiles and perform sufficient steps in the $\langle N \rangle$ direction, as shown in Figure 7.

Section Three. By positioning the group of robots in the third section as depicted in Figure 7a, stepping twice in the $\langle N \rangle$ direction will cause the topmost rows of the group of tiles to meet. This is repeated for every row by first stepping in the $\langle E \rangle$ direction, followed by two steps in the $\langle N \rangle$ direction. After every row has been made adjacent, repeating the step sequence $\langle N, E \rangle$ will output the robots from the funneling gadget, bringing together each column of robots and outputting the desired shape at the top of the gadget, (Figures 5c, 5d).

5 Future Work

Our work leads into a number of areas for future work. The first direction is to attempt to close the gaps between our upper bounds and our lower bounds for linear and general patterns. For lines, the goal is to close the $\Theta(\sqrt{k})$ gap between our upper and lower bounds, and with general shapes, we are interested in closing the gap for sparse shapes existing in large bounding boxes. Another direction is to consider how the unique placement requirement affects the required run-time. Without it, the $\Omega(\sqrt{k})$ lower bound no longer holds. Is it possible to achieve $O(n \log k)$ step complexity by placing different patterns at different locations? And if so, can this be done with a polynomial sized board? Finally, another interesting direction is to focus on pattern reconfiguration, similar to what [12] have looked at within the full tilt model. How fast can reconfiguration be done in the single-step model? Can reconfiguration be done quickly for general patterns and general shapes?

References

[1] Jose Balanza-Martinez, David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie, *Relocation with uniform external control in limited directions*, The 22^{nd} Japan Conference on Discrete and Computational Geometry, Graphs, and Games, JCDCGGG, 2019, pp. 39–40.

[2] Jose Balanza-Martinez, Timothy Gomez, David Caballero, Austin Luchsinger, Angel A. Cantu, Rene Reyes, Mauricio Flores, Robert T. Schweller, and Tim Wylie, *Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces*, Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA'20, SIAM, 2020, pp. 2625–2641.

[3] Jose Balanza-Martinez, Austin Luchsinger, David Caballero, Rene Reyes, Angel A. Cantu, Robert Schweller, Luis Angel Garcia, and Tim Wylie, *Full tilt: Universal constructors for general shapes with uniform external forces*, Proceedings of the 30^{th} Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'19, 2019, pp. 2689–2708.

[4] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Golnaz Habibi, and James McLurkin, *Reconfiguring massive particle swarms with limited, global control*, Algorithms for Sensor Systems (Berlin, Heidelberg) (Paola Flocchini, Jie Gao, Evangelos Kranakis, and Friedhelm Meyer auf der Heide, eds.), Springer Berlin Heidelberg, 2014, pp. 51–66.

[5] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Jarrett Lonsford, and Rose Morris-Wright, *Particle computation: complexity, algorithms, and logic*, Natural Computing **18** (2019), 6751–6756.

[6] 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*, 2017.

[7] Aaron T. Becker, Golnaz Habibi, Justin Werfel, Michael Rubenstein, and James McLurkin, *Massive uniform manipulation: Controlling large populations of simple robots with a common input signal*, 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nov 2013, pp. 520–527.

[8] Sheryl Manzoor, Samuel Sheckman, Jarrett Lonsford, Hoyeon Kim, Min Jun Kim, and Aaron T. Becker, *Parallel self-assembly of polyominoes under uniform control inputs*, IEEE Robotics and Automation Letters **2** (2017), no. 4, 2040–2047.

[9] Sylvain Martel, Samira Taherkhani, Maryam Tabrizian, Mahmood Mohammadi, Dominic de Lanauze, and Ouajdi Felfoul, *Computer 3d controlled bacterial transports and aggregations of microbial adhered nano-components*, Journal of Micro-Bio Robotics **9** (2014), no. 1, 23–28.

[10] Arne Schmidt, Sheryl Manzoor, Li Huang, Aaron T. Becker, and Sándor Fekete, *Efficient parallel self-assembly under uniform control inputs*, IEEE Robotics and Automation Letters (2018), 1–1.

[11] Yasuhiro Shirai, Andrew J. Osgood, Yuming Zhao, Kevin F. Kelly, and James M. Tour, *Directional control in thermally driven single-molecule nanocars*, Nano Letters **5** (2005), no. 11, 2330–2334, PMID: 16277478.

[12] Y. Zhang, X. Chen, H. Qi, and D. Balkcom, *Rearranging agents in a small space using global controls*, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 3576–3582.

New Results in Sona Drawing: Hardness and TSP Separation

Man-Kwun Chiu* Erik D. Demaine† Yevhenii Diomidov† David Eppstein‡ Robert A. Hearn§
Adam Hesterberg† Matias Korman¶ Irene Parada‖ Mikhail Rudoy**

In memoriam Godfried Toussaint (1944–2019)

Abstract

Given a set of point sites, a sona drawing is a single closed curve, disjoint from the sites and intersecting itself only in simple crossings, so that each bounded region of its complement contains exactly one of the sites. We prove that it is NP-hard to find a minimum-length sona drawing for n given points, and that such a curve can be longer than the TSP tour of the same points by a factor > 1.5487875. When restricted to tours that lie on the edges of a square grid, with points in the grid cells, we prove that it is NP-hard even to decide whether such a tour exists. These results answer questions posed at CCCG 2006.

1 Introduction

In April 2005, Godfried Toussaint visited the second author at MIT, where he proposed a computational geometric analysis of the "sona" sand drawings of the Tshokwe people in the West Central Bantu area of Africa. Godfried encountered sona drawings, in particular the ethnomathematical work of Ascher [1] and Gerdes [7], during his research into African rhythms. Together with his then-student Perouz Taslakian, we came up with a formal model of ***sona drawing*** of a set P of point ***sites*** — a closed curve drawn in the plane such that

1. wherever the curve touches itself, it crosses itself;
2. each crossing involves only two arcs of the curve;
3. exactly one site is in each bounded face formed by the curve; and
4. no sites lie on the curve or within its outside face.

Our first paper on sona drawings appeared at BRIDGES 2006 [5], detailing the related cultural practices, proving and computing combinatorial results and drawings, and posing several open problems. In early 2006, we brought these open problems to Godfried's Bellairs Winter Workshop on Computational Geometry, where a much larger group tackled sona drawings, resulting in a CCCG 2006 paper later the same year [4]. Next we highlight some of the key prior results and open problems as they relate to the results of this paper.

Sona vs. TSP. Every TSP tour can be easily converted into a sona drawing of roughly the same length: instead of visiting a site, loop around it, except for one site that we place slightly interior to the tour [5, Lemma 11]. Conversely, every sona drawing can be converted into a TSP tour of length at most a factor $\frac{\pi+2}{\pi} \approx 1.63661977$ larger [4, Theorem 12], settling [5, Open Problem 6]. Is this constant tight? The best previous lower bound was a four-site example proving a TSP/sona separation factor of $\frac{2}{3} + \frac{2\sqrt{3}}{9} \approx 1.05156685$ [5, Lemma 12]. In Section 2, we construct a recursive family of examples proving a much larger TSP/sona separation factor of $\frac{14+8\sqrt{2}+\pi(\sqrt{2}+1)}{8+4\sqrt{2}+\pi(\sqrt{2}+1)} \approx 1.54878753$. We also study L_1 and L_∞ metrics, where we prove that the worst-case TSP/sona separation factor is exactly 1.5.

Length minimization. The relation to TSP implies a constant-factor approximation algorithm for finding the minimum-length sona drawing on a given set of sites. But is this problem NP-hard? In Section 3, we prove NP-hardness for L_1, L_2, and L_∞ metrics, settling [5, Open Problem 5] and [4, Open Problem 4].

Grid drawings. The last variant we consider is when the sona drawing is restricted to lie along the edges of a unit-square grid, while sites are at the centers of cells of the grid. Not all point sets admit a grid sona drawing; however, if we scale the sites' coordinates by a factor of 3, then they always do [4, Proposition 10]. A natural remaining question [4, Open Problem 3] is which point sets admit grid sona drawings. In Section 4, we prove that this question is in fact NP-hard.

*Institut für Informatik, Freie Universität Berlin, Germany, chiumk@inf.fu-berlin.de. This work was supported in part by ERC StG 757609.

†CSAIL, Massachusetts Institute of Technology, USA, {edemaine,diomidov,achester}@mit.edu

‡Computer Science Department, University of California, Irvine, eppstein@uci.edu. This work was supported in part by the US National Science Foundation under grant CCF-1616248.

§bob@hearn.to

¶Tufts University, USA, matias.korman@tufts.edu

‖TU Eindhoven, The Netherlands, i.m.de.parada.munoz@tue.nl

**CSAIL, MIT, USA. Now at Google Inc.

2 Separation from TSP Tour

We first show an example which gives a large TSP/sona separation factor under the L_2 metric in the plane.

Theorem 1 *There exists a set of sites for which the length of the minimum-length TSP tour is* $\frac{14+8\sqrt{2}+\pi(\sqrt{2}+1)}{8+4\sqrt{2}+\pi(\sqrt{2}+1)} \approx 1.54878753$ *times the length of the minimum-length sona drawing.*

The full proof can be found in Appendix A.

Sketch of Proof. We construct a problem instance whose minimum-length TSP tour is longer than its minimum-length sona drawing by a factor within ε of $\frac{14+8\sqrt{2}+\pi(\sqrt{2}+1)}{8+4\sqrt{2}+\pi(\sqrt{2}+1)}$. Our construction is illustrated in Figure 1 and follows a fractal approach with ε^{-1} levels (for simplicity, we assume that ε^{-1} is an integer).

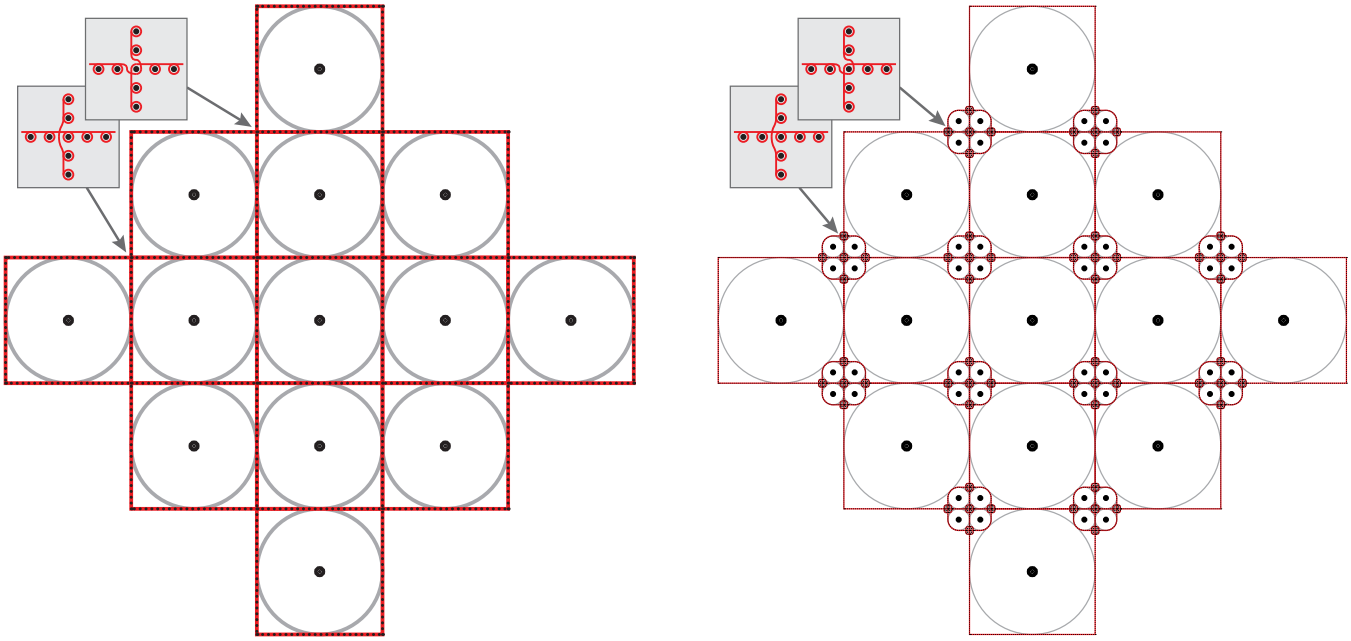
1. We start by defining a few auxiliary points:
   (a) The initial set of auxiliary points A_0 is the intersection between a slightly shifted integer lattice and the L_1 ball $B(0, \varepsilon^{-1})$ of radius ε^{-1} centered at the origin. That is, $A_0 = \left\{(\frac{2i+1}{2}, \frac{2j+1}{2}) : i, j \in \mathbb{Z}\right\} \cap \left\{(x, y) : |x| + |y| \leq \varepsilon^{-1}\right\}$. In Figure 1a, the auxiliary points are exactly the intersections of solid red lines.
   (b) Then, in Step i (starting with $i = 1$), for each auxiliary point $p \in A_{i-1}$, we add five points to A_i: p itself, and four new points at distance $\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$ from p in each of the four cardinal directions. Let $A = A_{\varepsilon^{-1}}$. By construction, we have $|A_i| = 5^i|A_0|$ and $|A_0| = 2\varepsilon^{-2} + O(\varepsilon^{-1})$. We note that set A contains auxiliary points (not sites). These points will not be part of the instance.
2. We now use the auxiliary points to create some sites (the isolated black points of Figure 1):
   (a) For any $i \geq 1$ we define set P_i of sites as follows: for each auxiliary point $q \in A_{i-1}$ we add the four sites whose x and y coordinates each differ from q by $\frac{1}{2}\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$. We note that all the added sites are distinct sites.
   (b) We define P_0 as the set of integer lattice points in $B(0, \varepsilon^{-1})$. Equivalently, for each auxiliary point $q \in A_0$ we add the sites whose x and y coordinates each differ from q by $\frac{1}{2}$, but we do not add sites that lie outside $B(0, \varepsilon^{-1})$, which affects $O(\varepsilon^{-1})$ sites (out of $\Omega(\varepsilon^{-2})$ sites of P_0). In this case, the sites created by different auxiliary points may lie in the same spot. In total, P_0 contains only one site per auxiliary point of A_0 (except for $O(\varepsilon^{-1})$ auxiliary points near the boundary). Thus, $|P_i| = 4 \cdot |A_{i-1}| = 4 \cdot 5^{i-1} \cdot |A_0|$ (for $i \geq 1$) and $|P_0| = |A_0| + O(\varepsilon^{-1}) = (2\varepsilon^{-2} + O(\varepsilon^{-1}))$.

   Let $P^{(1)} = P_0 \cup P_1 \cup \cdots \cup P_{\varepsilon^{-1}}$.
3. Next, we place additional sona sites packing line segments and/or curves. Whenever we pack any curve, we place sites spaced at a distance δ small enough that the length of the shortest path that passes within δ of all of them is within a factor $(1 - \varepsilon)$ of the length of the curve.
   (a) Solid lines as drawn in red in Figure 1a: for each $x \in \{i + \frac{1}{2} : -(\varepsilon^{-1}+1) \leq i \leq \varepsilon^{-1}$ and $i \in \mathbb{Z}\}$, we pack the vertical line segment with endpoints $(x, \varepsilon^{-1}+1-|x|)$ and $(x, |x|-\varepsilon^{-1}-1)$, and analogously with y for the horizontal line segments.
   (b) In Step i of the above recursive definition (starting with $i = 1$), when we create four new auxiliary points of A_i from a point $p \in A_{i-1}$, we also pack the boundary of the region within Euclidean distance $\frac{1}{2}\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$ of the square whose vertices are the four auxiliary points of A_i. Note that this boundary region forms a square with rounded corners as in Figure 1b. With these extra points we preserve the invariant that the auxiliary points are exactly the intersections of packed curves.

Let $P^{(2)}$ be the set of sites created in Step 3 in our construction, and $P = P^{(1)} \cup P^{(2)}$. This is a complete description of the construction.

In the full proof we show that the length of the packed curves is a $(1 + \varepsilon)$-approximation of the total length of the minimum-length sona drawing of P. Careful calculations then yield that the length of the minimum-length sona drawing is $(2\varepsilon^{-2} + O(\varepsilon^{-1}))\left(2 + \frac{4+\pi}{2\sqrt{2}-2}\right)$. We then argue that the minimum-length TSP has an additional length of $(2\varepsilon^{-2} + O(\varepsilon^{-1}))(2\sqrt{2} + 3)$. Thus, the TSP/sona separation factor for the construction is, ignoring lower-order terms, $\frac{2+\frac{4+\pi}{2\sqrt{2}-2}+2\sqrt{2}+3}{2+\frac{4+\pi}{2\sqrt{2}-2}} = \frac{14+\pi(\sqrt{2}+1)+8\sqrt{2}}{8+4\sqrt{2}+\pi(\sqrt{2}+1)} \approx 1.54878753$.

We have presented a construction for the L_2 metric in the plane showing that the ratio between the lengths of the minimum-length TSP and the minimum-length sona drawing can be strictly greater than 1.5. Our next result shows that this cannot be the case for the L_1 and L_∞ metrics in the plane.



(a) First step of our construction for $\varepsilon^{-1} = 2$: points of A_0 lie in the intersection of solid lines (packed segments). In the construction, P_1 contains thirteen sites (shown as black dots).

(b) Final construction for $\varepsilon^{-1} = 2$. Sets P_1, P_2, and P_3 are shown as black dots of varying sizes. Additional sites of $P^{(2)}$ pack lines and rounded squares nearby the auxiliary points of A_0, A_1, and A_2.

Figure 1: Recursive construction of sites requiring ≈ 1.54878753 factor shorter sona tour (drawn in red) compared to TSP tour (red plus doubled radius of each grey circle). All red lines have black sites sprinkled densely along them.

Theorem 2 *For the Manhattan (L_1) and the Chebyshev (L_∞) metrics, the minimum-length TSP tour for a set of sites P has length at most* 1.5 *times that of the minimum-length sona drawing for P. Moreover, this bound is tight for both metrics.*

Proof. The proof of the upper bound on the length of the minimum-length TSP tour follows the lines of the (unpublished) proof of [4, Theorem 12]. Let $P = \{p_1, \ldots, p_n\}$ be a set of n sites, $S(P)$ the minimum-length sona drawing for P, and $\mathrm{TSP}(P)$ the minimum-length TSP tour for P. The sona drawing $S(P)$ must have n bounded faces, each containing a site of P. Let f_i be the face of $S(P)$ containing the site p_i. In this proof, for an edge-weighted graph H, $|H|$ denotes the sum of the weights/lengths of all the edges of H. In particular, $|S(P)|$ denotes the length of the sona drawing $S(P)$.

For each site p_i, let $c(p_i)$ be the closest point in $S(P)$ to p_i and r_i the distance between p_i and $c(p_i)$. By the definition of $c(p_i)$, the open disk centered at p_i and with radius r_i does not intersect $S(P)$. This implies that the length of the boundary of f_i is at least the perimeter of a disk with radius r_i, that for both the L_1 and the L_∞ metrics is $8r_i$. That is, $|f_i| \leq 8r_i$. Moreover, the sum of the lengths of all the faces is $2|S(P)|$, so $|f_1| + \cdots + |f_n| < 2|S(P)|$ since we do not sum the length of the unbounded face.

We define a multigraph G whose vertex set is the union of the set of sites P, the set of vertices of $S(P)$, and $\{c(p_i) \in S : p_i \in P\}$. The edge set of G is the union of the set of edges of S and two parallel edges $\{p_i, c(p_i)\}$ for each $p_i \in P$. The weight of each edge is its length in the drawing. By the observations above, $|G| = |S(P)| + 2r_1 + \cdots + 2r_n \leq |S(P)| + |f_1|/4 + \cdots + |f_n|/4 < |S(P)| + |S(P)|/2 = 1.5|S(P)|$.

To obtain the desired upper bound on $|\mathrm{TSP}(P)|$ it remains to show that $|\mathrm{TSP}(P)| \leq |G|$. By construction, since $S(P)$ is Eulerian, so is G. An Euler tour of G defines a TSP tour for the vertices of G by skipping vertices that were already visited (as in the Christofides 1.5-approximation algorithm for TSP on instances where the distances form a metric space [3]). This TSP tour has length at most $|G|$ and can be shortcut so that it only visits the sites of P. By the triangle inequality, the length of the tour does not increase with these shortcuts. Thus, we have that $|\mathrm{TSP}(P)| \leq |G| < 1.5|S(P)|$.

The construction for the matching this bound is similar to the one in the proof of Theorem 1, but simpler. An illustration can be found in Figure 1a. For every $\varepsilon > 0$ we construct a set of sites P_ε (the set of TSP vertices/sona sites) such that $|\mathrm{TSP}(P_\varepsilon)| \geq (1.5 - \varepsilon)|S(P_\varepsilon)|$.

We fix $k = \lceil 1/(2\varepsilon) \rceil$. The set of sites P_ε includes every integer lattice point (x, y) such that $|x| + |y| \leq k$. Consider drawing Q resulting from the union of the axis-

aligned unit squares centered at these sites. It is easy to see, for example by rotating the construction, that so far we have added $(k+1)^2+k^2$ sites to P_ε and that the length of Q is $4(k+1)^2$. Straightforward computations show that $\frac{4(k+1)^2+(k+1)^2+k^2}{4(k+1)^2} = 5/4 + \frac{k^2}{4(k+1)^2} \geq 5/4 + \frac{1}{4(2\varepsilon+1)^2} = 1.5 - \varepsilon + \frac{\varepsilon^2(4\varepsilon+3)}{(2\varepsilon+1)^2} > 1.5 - \varepsilon$. Thus, a dense-enough packing of sites along Q yields the desired result. □

We next consider sona drawings on the sphere. By the definition of sona drawings in the plane, the unbounded face contains no sites. For the sphere we consider the following analogue: if there is a face that contains in its interior a half-sphere then this face contains no sites. Note that there is at most one such face. The following theorem shows a tight upper bound on the TSP/sona separation factor for drawings on the sphere. (We consider the usual metric inherited from the Euclidean metric in $\mathbb{R}^3$.)

Theorem 3 *For drawings on the sphere, the length of the minimum-length TSP tour for a set of sites P is at most 2 times the length of the minimum-length sona drawing for P. Moreover, this bound is tight.*

Proof. The proof of the upper bound on the length of the minimum-length TSP tour again follows the lines of the (unpublished) proof of [4, Theorem 12]. It only differs slightly from the first part of the the proof of Theorem 2. Using the same notation, in this case, the distance r_i between a site $p_i \in P$ and its closest point $c(p_i)$ in $S(P)$ corresponds to the length of the shortest arc on the great circle through p_i and $c(p_i)$. The open disk centered at p_i and with radius r_i is an open spherical cap that does not intersect $S(P)$. Assuming that the sphere has radius ρ, the boundary of this cap has length $2\pi\rho\sin(r_i/\rho)$. Since the face containing a site cannot contain a half-sphere in its interior we have that $0 \leq r_i/\rho \leq \pi/2$. The function $\sin(x)/x$ in the interval $0 \leq x \leq \pi/2$ is decreasing. Thus, $\rho/r_i \sin(r_i/\rho) \geq 2/\pi \sin(\pi/2) = 2/\pi$. This implies that $2\pi\rho\sin(r_i/\rho) \geq 4r_i$. Thus, the face f_i of $S(P)$ containing the site p_i has length $|f_i| \geq 4r_i$. Moreover, $|f_1| + \cdots + |f_n| \leq 2|S(P)|$. With the same arguments and defining the same multigraph as in the proof of Theorem 2 we obtain that $|\mathrm{TSP}(P)| \leq 2|S(P)|$.

The construction showing that this bound is tight places two sites on the north and south poles of the sphere and packs the equator densely with sites. Then the minimum-length sona drawing goes along the equator while the minimum-length TSP must reach both poles, yielding a $2-\varepsilon$ TSP/sona separation factor. □

3 Complexity of Length Minimization

In this section and Appendix B, we prove that finding a sona drawing of minimum length for given sites is NP-hard, even when the sites lie on a polynomially sized grid. The complexity of minimum-length sona drawing was posed as an open problem in 2006 by Damian et al. [4, Open Problem 4]. We use a reduction from the problem of finding a Hamiltonian cycle in a grid graph (a graph whose n vertices are a subset of the points in an integer grid, and whose edges are the unit-length line segments between pairs of vertices), proven NP-complete by Itai, Papadimitriou, and Szwarcfiter [8].

Let V be the set of n vertices in a hard instance for Hamiltonian cycle in grid graphs. If V is a YES instance, its Hamiltonian cycle forms a Euclidean traveling salesman tour with length exactly n. If it is a NO instance, the shortest Euclidean traveling salesman tour through its vertices has length at least 1 for every grid edge, and length at least $\sqrt{2}$ for at least one edge that is not a grid edge (as this is the shortest distance between grid points that are non-adjacent), so its total length is at least $n + \sqrt{2} - 1 \approx n + 0.414$. For the L_1 distance, the increase in length is larger, at least 1. Our reduction replaces each point of V by two points, close enough together to make the increase in length from converting a TSP to a sona drawing negligible with respect to this gap in tour length.

Theorem 4 *It is NP-hard to find a sona drawing for a given set of sites whose length is less than a given threshold L, for any of the L_1, L_2, and L_∞ metrics.*

4 Complexity of Grid Drawing Existence

While minimizing the length of sona drawings in general is hard, if we restrict the drawing to lie on a grid, then even determining the existence of a sona drawing is hard.

Given n sites at the centers of some cells in the unit-square grid, a ***grid sona drawing*** is a sona drawing whose edges are drawn as polygonal lines along the orthogonal grid lines (like orthogonal graph drawing).

We show that finding a grid sona drawing for a given set of sites is NP-hard by a reduction from Planar CNF SAT [9].

4.1 Construction

In this section we view the grid as a graph, thus by ***edge*** we mean a unit segment of a grid line, and by ***vertex*** we mean a grid vertex — these terms are distinct from "sona edge" etc. We say that an edge is either ***on*** or ***off*** according to as it belongs in the sona drawing. The subgraph of the grid that is on is the ***path graph***. Observe that two grid-adjacent sites always require the edge between them to be on; otherwise both would be in the same sona face (***connected***). Also, every vertex must have even degree in the path graph.

Here we are concerned with internal properties of the gadgets. Their exteriors are lined with unconnected

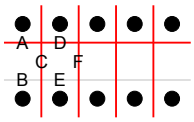


Figure 2: Wire gadget

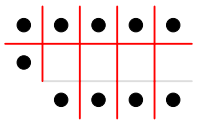


Figure 3: Constant gadget

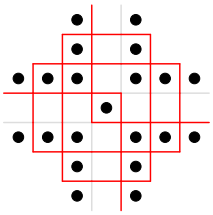


Figure 4: Turn / Split / Invert gadget

edges; we will show later how to connect them.

Wire. The ***wire*** gadget is shown in Figure 2. One of edges A and B must be on, otherwise two sites would be connected. Assume without loss of generality that A is on. Now, suppose C is off. Then D must be too, to preserve even vertex degree. Then, B and E must both be on to prevent sites from being connected, but this is impossible with C off. Therefore C and D are on. The same reasoning shows that the entire line A, D, etc. is on, as well as edges C, F, etc. Then E must be off to prevent an empty face, and thus the entire line B, E, etc. is off. The wire thus has two states: the upper line can be on and the lower off, or vice-versa. We can extend a wire as long as necessary. An unconnected wire end serves as a variable.

In Figures 2 through 5, all marked edge states (red for on, gray for off) are forced by the indicated wire states. These marks were generated by computer search, but are easy to verify by local analysis.

Constant. The gadget shown in Figure 3 forces the attached wire to be in the up state: the edge between the two left sites must be on, forcing the rest.

Turn / Split / Invert. The gadget shown in Figure 4 is multi-purpose. If we view the left wire as the input, then the upper and lower outputs represent turned signals, and the right output represents an inverted signal. (Unused outputs can be left unattached, thus unconstrained.)

Any wire state forces all the others. Given that the left wire is in the up state, suppose the right wire is also up. Then the top wire and bottom wire must be in the same left/right state, otherwise we will have degree-three vertices in the middle. But this would leave the central site connected to another site, so the right wire is forced down. Then, if the (top, bottom) wires are not in the (left, right) states, again the central site will be connected to another one. (This figure contains multiple loops, but these will be eliminated in the final configuration by adding more edges.)

OR. Figure 5 shows the OR gadget. The upper wire is interpreted as an output, with the left state representing true; the other wires are inputs, with true represented as down on the left wire and up on the right wire. (We can easily adjust truth representations between gadgets with inverters.) If either input is true, then the output may be set true, as shown. If both inputs are false, the output may be set false. Figure 5e shows that setting the output to true when both inputs are false is not possible: all marked edge states are forced by the wire properties, but two sites are left connected.

4.2 Hardness

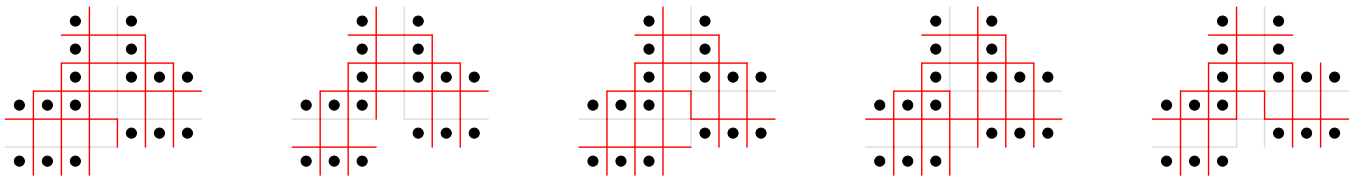
Theorem 5 *It is NP-hard to find a grid sona drawing for a given set of sites at the centers of grid cells.*

Proof. Given a CNF Boolean formula with a planar incidence graph, we connect the above gadgets to represent this graph: unconstrained wire ends represent variables, and are connected to splitters and inverters to reach clause constructions. A clause is implemented with chained OR gadgets, with the final output constrained to be true with a constant gadget. By the gadget properties described above, we will be able to consistently choose wire states if and only if the formula is satisfiable.

We must still show that all edges can be joined together into a single closed loop, while retaining the sona properties. Our basic strategy for connecting loose ends is to border each gadget with "crenellations", as shown in Figure 6. This figure also shows how to pass pairs of path segments across a wire without affecting its internal properties, which we will use to help form a single loop.

Adding crenellations to the other gadgets is straightforward, and we defer explicit figures to Appendix C, with one exception. (The crenellations do add a parity constraint when wiring gadgets together; we show in the appendix how to shift parity.) When we use the gadget in Figure 4 to turn a wire, it will be useful to use the crenellated version in Figure 7. With the connections to other gadgets on the left and top, the right and bottom portions are unconstrained. We can place edges as shown, so that they leave the gadget identically regardless of which state it is in. Then, all paths entering from the left or the top leave on the bottom or the right as loose edges, except that in Figure 7a, one path connects the left to the top. If we connect a right turn to the top port, this path will also terminate in an unconnected edge. If every wire contains a left turn and matching right turn, then every path in the sona graph must end in two unconnected edges in turn gadgets, because there are no internal loops in any of the gadgets.

The space occupied by the loose ends of a turn lies either in an internal face of the wiring graph, or on its exterior. We route the interior ends to pass-through pairs as shown in Figure 6, so all unconnected edges



(a) false + true → true (b) true + true → true (c) true + false → true (d) false + false → false (e) Bad state

Figure 5: OR gadget

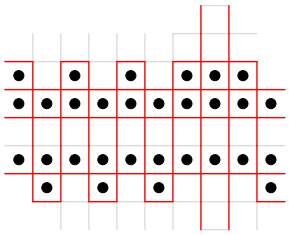
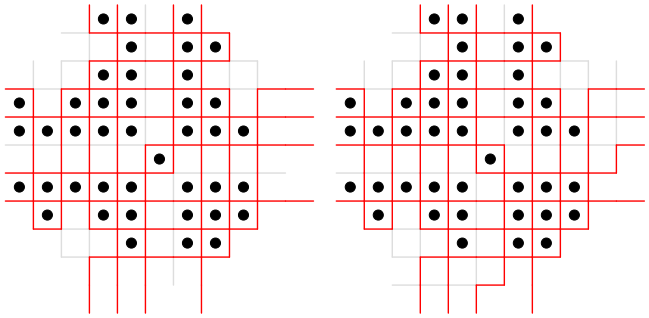


Figure 6: Wire gadget with crenellations and pass-through



(a) State 1 (b) State 2

Figure 7: Crenellated turn

wind up on the outer border of the graph. Because the terminal edges are placed identically in Figures 7a and 7b, we can plan their routing without knowing the wire states. As a result, we can place additional sites as required for the property that a single site lies in each internal sona face. (We can lengthen the wires as needed to create additional routing space in the internal faces.)

Now we are in a state where all paths end on the exterior of the construction. If we join these paths together without crossing, the number of extra sites needed in the outer face is just the number of paths. We place that many sites in a widely spaced grid (spacing proportional to number of paths) surrounding the inner construction. Then, we can complete the path greedily by repeatedly connecting one outer path end to one of its neighboring path ends, surrounding one of the added sites. Only one of its two neighboring path ends can come from the same path, so there's always another one to connect to. The wide grid spacing of the outer sites means there is always room to route the connection. □

Acknowledgments

Thanks to Godfried Toussaint for introducing us (and computational geometry) to sona drawings. This research was initiated during the Virtual Workshop on Computational Geometry held March 20–27, 2020, which would have been the 35th Bellairs Winter Workshop on Computational Geometry co-organized by E. Demaine and G. Toussaint if not for other circumstances. We thank the other participants of that workshop for helpful discussions and providing an inspiring atmosphere.

References

[1] Marcia Ascher. *Mathematics Elsewhere: An Exploration of Ideas Across Cultures*. Princeton University Press, 2002.

[2] Molly Baird, Sara C. Billey, Erik D. Demaine, Martin L. Demaine, David Eppstein, Sándor Fekete, Graham Gordon, Sean Griffin, Joseph S. B. Mitchell, and Joshua P. Swanson. Existence and hardness of conveyor belts. Electronic preprint arXiv:1908.07668, 2019. URL: https://arXiv.org/abs/1908.07668.

[3] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[4] Mirela Damian, Erik D. Demaine, Martin L. Demaine, Vida Dujmović, Dania El-Khechen, Robin Flatland, John Iacono, Stefan Langerman, Henk Meijer, Suneeta Ramaswami, Diane L. Souvaine, Perouz Taslakian, and Godfried T. Toussaint. Curves in the sand: Algorithmic drawing. In *Proceedings of the 18th Annual Canadian Conference on Computational Geometry (CCCG 2006)*, pages 11–14, Kingston, Ontario, August 2006. URL: https://cccg.ca/proceedings/2006/cccg4.pdf.

[5] Erik D. Demaine, Martin L. Demaine, Perouz Taslakian, and Godfried T. Toussaint. Sand drawings and Gaussian graphs. *Journal of Mathematics and The Arts*, 1(2):125–132, June 2007. Originally at BRIDGES 2006. doi:10.1080/17513470701413451.

[6] Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke. Problem 33: Sum of Square Roots. In *The Open Problems Project*. Smith College, September 19 2017. URL: https://cs.smith.edu/~jorourke/TOPP/P33.html.

[7] Paulus Gerdes. The 'sona' sand drawing tradition and possibilities for its educational use. In *Geometry From Africa: Mathematical and Educational Explorations*, pages 156–205. The Mathematical Association of America, 1999.

[8] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982. doi:10.1137/0211056.

[9] David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. URL: https://doi.org/10.1137/0211025, doi:10.1137/0211025.

[10] Joseph O'Rourke. Advanced problem 6369. *American Mathematical Monthly*, 88(10):769, 1981. doi:10.2307/2321488.

A Separation from TSP Tour under the L_2 Metric

Theorem 1 *There exists a set of sites for which the length of the minimum-length TSP tour is* $\frac{14+8\sqrt{2}+\pi(\sqrt{2}+1)}{8+4\sqrt{2}+\pi(\sqrt{2}+1)} \approx 1.54878753$ *times the length of the minimum-length sona drawing.*

Proof. We construct a problem instance whose minimum-length TSP tour is longer than its minimum-length sona drawing by a factor within ε of $\frac{14+8\sqrt{2}+\pi(\sqrt{2}+1)}{8+4\sqrt{2}+\pi(\sqrt{2}+1)}$. Our construction is illustrated in Figure 1 and follows a fractal approach with ε^{-1} levels (for simplicity, we assume that ε^{-1} is an integer).

1. We start by defining a few auxiliary points:
   (a) The initial set of auxiliary points A_0 is the intersection between a slightly shifted integer lattice and the L_1 ball $B(0, \varepsilon^{-1})$ of radius ε^{-1} centered at the origin. That is, $A_0 = \left\{(\frac{2i+1}{2}, \frac{2j+1}{2}) : i, j \in \mathbb{Z}\right\} \cap \left\{(x, y) : |x| + |y| \leq \varepsilon^{-1}\right\}$. In Figure 1a, the auxiliary points are exactly the intersections of solid red lines.
   (b) Then, in Step i (starting with $i = 1$), for each auxiliary point $p \in A_{i-1}$, we add five points to A_i: p itself, and four new points at distance $\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$ from p in each of the four cardinal directions. Let $A = A_{\varepsilon^{-1}}$. By construction, we have $|A_i| = 5^i|A_0|$ and $|A_0| = 2\varepsilon^{-2} + O(\varepsilon^{-1})$. We note that set A contains auxiliary points (not sites). These points will not be part of the instance.
2. We now use the auxiliary points to create some sites (the isolated black points of Figure 1):
   (a) For any $i \geq 1$ we define set P_i of sites as follows: for each auxiliary point $q \in A_{i-1}$ we add the four sites whose x and y coordinates each differ from q by $\frac{1}{2}\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$. We note that all the added sites are distinct sites.
   (b) We define P_0 as the set of integer lattice points in $B(0, \varepsilon^{-1})$. Equivalently, for each auxiliary point $q \in A_0$ we add the sites whose x and y coordinates each differ from q by $\frac{1}{2}$, but we do not add sites that lie outside $B(0, \varepsilon^{-1})$, which affects $O(\varepsilon^{-1})$ sites (out of $\Omega(\varepsilon^{-2})$ sites of P_0). In this case, the sites created by different auxiliary points may lie in the same spot. In total, P_0 contains only one site per auxiliary point of A_0 (except for $O(\varepsilon^{-1})$ auxiliary points near the boundary). Thus, $|P_i| = 4 \cdot |A_{i-1}| = 4 \cdot 5^{i-1} \cdot |A_0|$ (for $i \geq 1$) and $|P_0| = |A_0| + O(\varepsilon^{-1}) = (2\varepsilon^{-2} + O(\varepsilon^{-1}))$.

   Let $P^{(1)} = P_0 \cup P_1 \cup \cdots \cup P_{\varepsilon^{-1}}$.
3. Next, we place additional sona sites packing line segments and/or curves. Whenever we pack any curve, we place sites spaced at a distance δ small enough that the length of the shortest path that passes within δ of all of them is within a factor $(1 - \varepsilon)$ of the length of the curve.
   (a) Solid lines as drawn in red in Figure 1a: for each $x \in \{i + \frac{1}{2} : -(\varepsilon^{-1}+1) \leq i \leq \varepsilon^{-1}$ and $i \in \mathbb{Z}\}$, we pack the vertical line segment with endpoints $(x, \varepsilon^{-1}+1-|x|)$ and $(x, |x|-\varepsilon^{-1}-1)$, and analogously with y for the horizontal line segments.
   (b) In Step i of the above recursive definition (starting with $i = 1$), when we create four new auxiliary points of A_i from a point $p \in A_{i-1}$, we also pack the boundary of the region within Euclidean distance $\frac{1}{2}\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$ of the square whose vertices are the four auxiliary points of A_i. Note that this boundary region forms a square with rounded corners as in Figure 1b.

With these extra points we preserve the invariant that the auxiliary points are exactly the intersections of packed curves.

Let $P^{(2)}$ be the set of sites created in Step 3 in our construction, and $P = P^{(1)} \cup P^{(2)}$. This is a complete description of the construction.

We now find the minimum-length sona drawing of P. Each pair of consecutive points in a packed curve must be in a separate sona region, so any sona drawing must pass between them; in particular, any sona drawing must pass within δ of each of them, and so the length of any valid sona drawing is at least $1-\varepsilon$ times the length of the packed curves. Also, there's a valid sona drawing that's at most $1+\varepsilon$ times the length of the packed curves: follow all the packed curves exactly, adding small loops around the sona sites of the packed curves as necessary (loops small enough to lengthen the curve by a factor of at most $1+\varepsilon$). The graph of packed curves is Eulerian (because it's defined as a union of boundaries of regions, which are cycles), so the TSP tour can follow an Eulerian circuit through it. At an intersection of packed curves, we have two options for the sona drawing (as shown in the inset images of Figure 1). We can have one sona path cross over the other in the Eulerian circuit (by including every site of the packed curve in a small loop). Alternatively, we can have one sona path cross over the other in two places p and q at the intersection, and leaving one sona site of the packed curve out of a small loop to be the sona site of the extra region between p and q. In either case, we conclude that there is a valid sona path that follows an Eulerian circuit of the packed curves within $(1+\varepsilon)$. Note that, although our description focused in the sites of $P^{(2)}$, this is a valid sona tour for P since the sites of $P^{(1)}$ lie in different faces.

The total length of the packed curves is hence a $(1+\varepsilon)$-approximation of the total length of the minimum-length sona drawing.

The total length of the packed segments of $P^{(2)}$ (the square lattice) is $4\varepsilon^{-2} + O(\varepsilon^{-1})$, since the area of the region $|x| + |y| < \varepsilon^{-1}$ and the number of lattice points in it are each $2\varepsilon^{-2} + O(\varepsilon^{-1})$.

Now we bound the length of the packed curves (rounded squares). In Step i of the construction (starting with $i = 1$), we added a packed curve that is the boundary of the region within Euclidean distance $\frac{1}{2}\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$ of a square of side length $\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$ (with a total length of $(4+\pi)\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$).

Recall that we added one such curve for each of the points of A_{i-1} and that $|A_i| = 5^{i-1}(2\varepsilon^{-2} + O(\varepsilon^{-1}))$. Thus, the total length of the sona drawings introduced at Step i is $(4+\pi)\left(\frac{1}{1+\sqrt{2}}\right)^{2i} 5^{i-1}(2\varepsilon^{-2} + O(\varepsilon^{-1}))$.

For ε small, this series is well-approximated by an infinite geometric series with sum $(2\varepsilon^{-2} + O(\varepsilon^{-1}))\left(\frac{4+\pi}{(1+\sqrt{2})^2 \cdot \left(1-\frac{5}{(1+\sqrt{2})^2}\right)}\right) = (2\varepsilon^{-2} + O(\varepsilon^{-1}))\left(\frac{4+\pi}{2\sqrt{2}-2}\right)$, and adding in the length of the packed segments of $P^{(2)}$ (the square lattice) gives $(2\varepsilon^{-2} + O(\varepsilon^{-1}))\left(2 + \frac{4+\pi}{2\sqrt{2}-2}\right)$.

We have approximated the minimum length of a valid sona drawing; now we approximate the minimum length of a TSP tour.

Any TSP tour must also come within δ of every point on every packed curve, which requires a length at least $(2\varepsilon^{-2} + O(\varepsilon^{-1}))\left(2 + \frac{4+\pi}{2\sqrt{2}-2}\right)$ as above. Also, the TSP tour must visit each site of $P^{(1)}$. We observe some properties of this set:

- Set $P^{(1)}$ is defined so that sites are far from each other. Specifically, the Euclidean ball centered at any site $p \in P_i$ of radius $r_i = \frac{1}{2}\left(\frac{1}{1+\sqrt{2}}\right)^{2i}$ does not contain other sona sites. This means that we must include at least $2r_i$ in the length of the TSP tour for each point in P_i, for the part of the tour that passes from the boundary of this ball to P_i and then back to the boundary.
- There are $4 \cdot 5^{i-1}(2\varepsilon^{-2} + O(\varepsilon^{-1}))$ sites in P_i (for $i \geq 1$) and $(2\varepsilon^{-2} + O(\varepsilon^{-1}))$ sites in P_0.

When ε tends to zero, the additional length needed in the TSP tour is

$$\begin{aligned}
& (2\varepsilon^{-2} + O(\varepsilon^{-1}))(1 + \sum_{i\geq 1} 2r_i \cdot 4 \cdot 5^{i-1}) \\
= \; & (2\varepsilon^{-2} + O(\varepsilon^{-1}))(1 + \frac{4}{5}\sum_{i\geq 1}\left(\frac{5}{3+2\sqrt{2}}\right)^i) \\
= \; & (2\varepsilon^{-2} + O(\varepsilon^{-1}))(1 + \frac{4}{3+2\sqrt{2}} \cdot \frac{1}{1-\frac{5}{3+2\sqrt{2}}}) \\
= \; & (2\varepsilon^{-2} + O(\varepsilon^{-1}))(1 + \frac{4}{2\sqrt{2}-2}) \\
= \; & (2\varepsilon^{-2} + O(\varepsilon^{-1}))(2\sqrt{2}+3).
\end{aligned}$$

So, the total length of the TSP tour is at least $(2\varepsilon^{-2} + O(\varepsilon^{-1}))\left(2 + \frac{4+\pi}{2\sqrt{2}-2} + 2\sqrt{2} + 3\right)$. Hence the ratio of the length of the TSP tour to the length of the sona drawing is, ignoring lower-order terms, $\frac{2+\frac{4+\pi}{2\sqrt{2}-2}+2\sqrt{2}+3}{2+\frac{4+\pi}{2\sqrt{2}-2}} = \frac{14+\pi(\sqrt{2}+1)+8\sqrt{2}}{8+4\sqrt{2}+\pi(\sqrt{2}+1)} \approx 1.54878753$. □

B Complexity of Length Minimization

Theorem 4 *It is NP-hard to find a sona drawing for a given set of sites whose length is less than a given threshold L, for any of the L_1, L_2, and L_∞ metrics.*

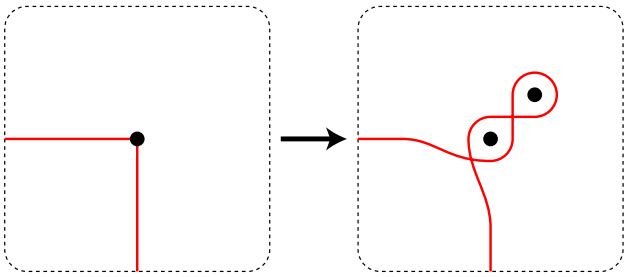


Figure 8: Local modifications to convert a grid Hamiltonian cycle into a short sona drawing for a set of doubled sites

Proof. Let V be the set of n vertices in a hard instance for finding a Hamiltonian cycle in grid graphs. We may form a hard instance of the minimum-length sona drawing problem for L_1 or L_2 distances by replacing each vertex in V by a pair of sites, one at the original vertex position and the other at distance less than ε from it, where $\varepsilon = \Theta(1/n)$ is chosen to be small enough that $4n\varepsilon < \sqrt{2} - 1$. We set $L = n + 2n\varepsilon$. For L_∞ distance, we use a hard instance for L_1 distance, rotated by 45°.

If V is a yes-instance for Hamiltonian cycle, let C be a Hamiltonian cycle of length n for V. We may form a sona drawing of length less than L by modifying C within a neighborhood of each pair of sites so that, for all but one of these pairs, it makes two loops, one surrounding each site (Figure 8), and so that for the remaining pair it makes one loop around one of the two sites and surrounds the other point by the face formed by C itself. In this way, each face of the modified curve surrounds a single site of our instance. Each of these local modifications to C may be performed using additional length less than 2ε, so the total length of the resulting sona drawing is less than L.

If V is a NO instance for Hamiltonian cycle, let C be any sona drawing for the resulting instance of the minimum-length sona drawing problem. Then C must pass between each pair of sites in the instance, and by making a local modification of length at most 2ε near each pair, we can cause it to touch the point in the pair that belongs to V itself. Thus, we have a curve of length $|C| + 2n\varepsilon$ touching all points of V. Because V is a NO instance, the length of this curve must be at least $n + \sqrt{2} - 1$, from which it follows that the length of C is at least $n + \sqrt{2} - 1 - 2n\varepsilon \geq L$. □

By scaling the sites by a factor of $\Theta(1/\varepsilon) = O(n)$ we may obtain a hard instance of the minimum-length sona drawing problem in which all sites lie in an integer grid whose bounding box has side length $O(n^2)$.

It is possible to represent a minimum-length sona drawing combinatorially, as a ***conveyor belt*** [2] formed by bitangents and arcs of infinitesimally small disks centered at each site, and to verify in polynomial time that a representation of this form is a valid sona drawing. However, this does not suffice to prove that the decision version of the minimum-length sona drawing problem belongs to NP. The reason is that, when the sites have integer coordinates, the limiting length of a sona drawing, represented combinatorially in this way, is a sum of square roots (distances between pairs of given points) and we do not know the computational complexity of testing inequalities involving sums of square roots [6, 10]. (Euclidean TSP has the same issue.)

C Crenellations for Grid Drawing

Figures 9, 10, and 11 show how to add crenellations to the Constant gadget, an unconstrained wire end (variable), and the OR gadget, respectively. The crenellated Split / Invert is the same as in Figure 7, extended in the obvious way for ports that are used. In no case do the crenellations affect the internal properties described in the main text; these figures simply show that it is possible to add the crenellations appropriately.

As mentioned in the main text, the crenellations do add a parity constraint when connecting gadgets with wires; we can no longer make wires of arbitrary length, but must match the crenellations to the gadgets at each end. In order to do that we need one additional gadget, an inverting turn, shown in Figure 12. Unlike in Figure 7, the wire state is switched during the turn. Observe that in Figure 7, turning does not change crenellation parity, but the straight-through path, which would invert if not terminated, does change crenellation parity. The inverting turn also does not change crenellation parity. Therefore, to change the crenellation parity of a wire, we can invert it (straight through), changing the parity, and add a sequence left inverting turn, right turn, right turn, left turn to restore the original line of the wire.

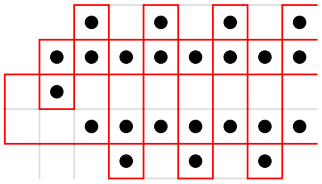
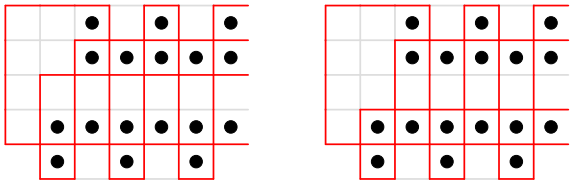
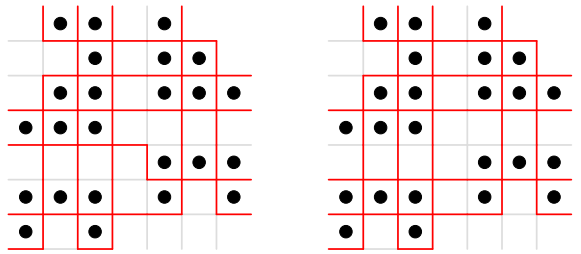


Figure 9: Crenellated Constant gadget



(a) State 1 (b) State 2

Figure 10: Crenellated unconstrained wire end



(a) false + true → true (b) true + true → true

(c) true + false → true (d) false + false → false

Figure 11: Crenellated OR gadget

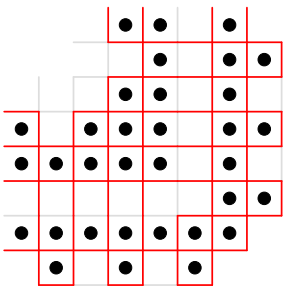


Figure 12: Crenellated inverting Turn gadget

Minimizing The Maximum Distance Traveled To Form Patterns With Systems of Mobile Robots

Jared Coleman* Evangelos Kranakis† Oscar Morales-Ponce‡ Jaroslav Opatrny§ Jorge Urrutia¶
Birgit Vogtenhuber‖

Abstract

In the pattern formation problem, robots in a system must self-coordinate to form a given pattern, regardless of translation, rotation, uniform-scaling, and/or reflection. In other words, a valid final configuration of the system is a formation that is *similar* to the desired pattern. While there has been no shortage of research in the pattern formation problem under a variety of assumptions, models, and contexts, we consider the additional constraint that the maximum distance traveled among all robots in the system is minimum. Existing work in pattern formation and closely related problems are typically application-specific or not concerned with optimality (but rather feasibility). We show the necessary conditions any optimal solution must satisfy and present a solution for systems of three robots. Our work also led to an interesting result that has applications beyond pattern formation. Namely, a metric for comparing two triangles where a distance of 0 indicates the triangles are similar, and 1 indicates they are *fully dissimilar*.

1 Introduction

While distributed systems have clear advantages over centralized ones, their complexity has stunted their potential in the mobile robotics market. Where distributed systems are cheap to build, scalable, and fault-tolerant in theory, they are extremely difficult to properly design in practice [14]. In this paper, we present results from a study on pattern formation, a common problem in distributed robotics. In the pattern formation problem, a system of mobile robots on the plane must move to form a given pattern. While this problem has been studied extensively, we consider the additional constraint that the maximum distance traveled among all robots must be minimum. For the purpose of this paper, we call solutions that satisfy this constraint *optimal*.

The main goal of this study is to develop a theoretical understanding of the pattern formation problem. In this study, we make contributions to establishing this baseline and, in doing so, make many interesting observations about properties and limitations for patterns and the systems that form them.

Our Contributions. The goal of this study is to develop a theoretical understanding of the min-max traversal pattern formation problem. To do so, we first explore the necessary conditions that any optimal solution must satisfy. For example, we prove in Section 4 (Lemma 1) that for any optimal solution, at least three robots must travel exactly the maximum distance. Notice that for systems of three robots, this means all three robots must move exactly the same distance, regardless of the pattern they must form. Clearly, the three-robot case is an important lower bound for the general case and is therefore the primary focus of this study. In Section 6, we present an algorithm for computing the optimal solution for systems of three robots. While not directly applicable, the three-robot solution has important implications on systems of many robots. In Section 9, we discuss these implications in further detail.

Our work on systems of three robots also yielded a surprising, but profound result. In Section 7, we prove that by modifying the aforementioned algorithm slightly, we can use it as a metric for measuring the similarity between two arbitrary triangles. This has potential applications beyond pattern formation for mobile robotic systems, like computational geometry and computer vision.

Models. In this paper, we are interested in the globally optimal solution to the pattern formation problem. Different models, however, may or may not be able to compute the optimal solution. In this section, we briefly discuss various models used in related literature and their implications on the pattern formation problem.

*California State University, Long Beach, jared.coleman@student.csulb.edu
†Carleton University, School of Computer Science, Ottawa, Ontario K1S 5B6, Canada, Research supported in part by NSERC Discovery grant kranakis@scs.carleton.ca
‡California State University, Long Beach, oscar.morales-ponce@csulb.edu
§Concordia University, Department Computer Science and Engineering, Montréal QC H3G 1M8 Canada opatrny@cs.concordia.ca
¶Instituto de Matematicas, UNAM, Mexico City, Mexico urrutia@matem.unam.mx
‖Institute of Software Technology, University of Technology, Graz, Austria bvogt@ist.tugraz.at

All models discussed in this paper follow the *look, compute, move* execution cycle. In the *look* phase, each robot observes the position of all other robots in the system (either globally or relative to their own local coordinate frame). Then, robots *compute* a solution and *move* some distance towards it. We also assume that, in each cycle, all robots move the same distance δ toward their destination unless they reach it, in which case they move some distance less than δ.

In accordance with related literature, we consider whether robots in the system are globally coordinated, oblivious, oriented, and/or synchronous. Robots are *globally coordinated* if they have access to a global coordinate system, otherwise they are said to be *locally coordinated*. Robots are *oblivious* if they do not have access to previous states of the system. In oblivious models, a solution must be computed using only a snapshot of the system at a given time. Robots are *oriented* if they have a common sense of direction (i.e. North, South, East, and West), otherwise they are *unoriented*. Robots are synchronous if they start each phase of their *look, compute, move* cycles at the same time (according to some global clock). In this paper, we also assume synchronous robots move at the same speed.

It has been shown that asynchronous and oblivious robots cannot form any arbitrary pattern (Theorem 3.1 in [15]). It has also been shown that locally coordinated, synchronous robots cannot form any arbitrary pattern (even sub-optimally) [15], but that locally coordinated, asynchronous robots can as long as they are oriented [10] due to possible symmetry in the initial configuration of robots. We assume robots are in general position and therefore do not consider the special case where robots are symmetric. Note that for any special case where robots are synchronous with each other, we can perturbate each robot's position by some small arbitrarily random amount to break symmetry. Table 1 is a summary of which models can and cannot form patterns optimally or sub-optimally for systems of three robots.

In this paper, we show that our solution for systems of three robots is valid under all globally coordinated, synchronous models and under the locally coordinated, oblivious, synchronous, and oriented model.

Notation. For any system of n robots, we denote their initial positions by $R = (r_0, r_1, \ldots, r_{n-1})$ (robot i is at position r_i). We define a pattern to be a sequence of distinct points on the plane and use capital letters, like P and S, to denote them. Lower-case letters and subscript indices are used to denote the elements of the sequence. For example, p_i is the i^{th} element of P. Sets of sequences of distinct points on the plane (e.g. sets of patterns) are denoted in calligraphic font, for example $\mathcal{P}$ and $\mathcal{S}$. Elements of these sets are denoted with their non-calligraphic equivalent and a superscript index. For example, S^i is the i^{th} element of $\mathcal{S}$ and s^i_j is the j^{th} element of S^i.

The number of elements in a sequence P, or its length, is denoted by $|P|$. Two sequences P and Q are equivalent, or $P = Q$, if and only if $|P| = |Q|$ and $p_i = q_i$ for $0 \leq i < |P|$. We say P and Q are similar, or $P \sim Q$ if and only if there exists some translation, rotation, uniform scaling, and/or reflection of any permutation of P that is equivalent to Q. P and Q are rigidly similar, or $P \overset{*}{\sim} Q$ if and only if there exists some translation, rotation, and/or uniform scaling of P, say P', such that $P' = Q$. Observe that $P \overset{*}{\sim} Q \Rightarrow P \sim Q$, but $P \sim Q \not\Rightarrow P \overset{*}{\sim} Q$.

Let $C(p, r)$ be the circle centered at p with radius r and $D(p, r)$ be the closed disk with center p and radius r. Also, let $d(u, v)$ be the Euclidian distance between points u and v.

Outline. This paper is organized as follows. First, we formally introduce the problem statement in Section 2 and discuss related work in Section 3. Then, we discuss the necessary conditions any optimal solution must satisfy in Section 4. In Section 5, we introduce Replication, a tool we use in Section 6 to show that our main contribution, an optimal solution for systems of three robots, is in fact optimal. In Section 7, we present a metric based on the optimal solution for systems of three robots. In Section 8, we discuss some properties of systems of three robots and the patterns they can form. Finally, Section 9 concludes this study with a discussion about future work and the significance of our contributions toward a theoretical understanding of the pattern formation problem.

2 Problem Statement

Consider a system of n robots with initial positions $R = (r_0, r_1, ..., r_{n-1})$. The trajectory of robot i is defined as a continuous function $f_i(t)$ for all $t \geq 0$. A strategy A defines a trajectory for every robot. Given a pattern P, we say that the strategy A is *valid* if there exists a time t such that the robots' positions are similar to P. Otherwise the strategy is *invalid*. To simplify notation we say robots that use a valid strategy *form* P. Let $t(A)$ be the earliest time at which the robots form P using strategy A. The distance that each robot traverses is defined as $d^A_i = \int_0^{t(A)} f^A_i(t)dt$.

In this study we are interested in a strategy that minimizes the maximum distance any robot traverses to form the desired pattern:

Problem 1 (Min-Max Traversal Pattern Formation) *Given a system of $n \geq 3$ robots with initial positions R and a pattern P, determine the minimum d^* for which there exists a valid strategy for forming P such that ev-*

Globally Coordinated	Oblivious	Synchronous	Oriented	Pattern Formable	
Yes	-	Yes	-	Optimal	Corollary 9,Theorem 7
	No	No	-	Valid	[15]
No	Yes	Yes	Yes	Optimal	Corollary 9,Theorem 7
		-	No	Impossible	[15]
	No	Yes	-	Optimal	Corollary 9
		No	Yes	Valid	[10]
			No	Impossible	[10]

Table 1: A globally optimal pattern is only formable in the general case under some models. Under some models, a valid sub-optimal formation can always be formed while under others, valid formations are not formable *at all* in the general case. Note that the results reported in this paper are only valid for systems of three robots.

ery robot travels at most distance d^*. Formally:

$$d^* = \min_{\forall A \in \mathcal{A}} (\max_{0 \leq i < n} (d_i^A))$$

where $\mathcal{A}$ is the set of all valid strategies.

3 Related Work

The pattern formation problem has been studied extensively under a variety of assumptions, models, and contexts. Many researchers use the pattern formation problem to study the algorithmic limitations of autonomous mobile robots [10, 15]. It has been shown, for example, that systems of synchronous robots with initially symmetric positions cannot form any geometric pattern [15] but that systems of asynchronous robots with compasses (A global sense of North/South and East/West) can [10]. We mitigate the problems that symmetry introduces to the pattern formation problem by assuming robots initial positions are random, and the probability of exact symmetry approaches zero. Since we are interested in finding any *theoretically* optimal solution for the general case, a feasibility discussion is out of the scope for this paper and left as future work.

Researchers have proposed solutions for many variants of the pattern formation problem. For example, it has been shown that it is possible to form a uniform circle (one such that the distance between neighboring robots on the circle is equal) for any system of robots arbitrarily deployed on the plane [8]. Other variations of the pattern formation that have been studied include gathering on a ring [11], point-convergence [4], and forming a series of patterns in succession [6]. There has also been work in variations of these problems where robots have visibility constraints, that is, they can only see other robots in the system if they are within a given distance [3, 9, 5]. Various methods and solutions for bio-inspired pattern formation are reviewed in [13]. When the destination positions are known, the pattern formation is reduced to robot-destination matching. There are many available solutions for these kinds of variants of the problem that guarantee a variety of different properties (i.e. no collision, minimum total distance traveled, etc.) [2]. Solutions typically involve a combinatorial optimization algorithm for the assignment problem, like the Hungarian Algorithm [12]. The quantity and variety of the literature reflects the seemingly unlimited variants and applications of the pattern formation problem. There is, however, no unifying theory that ties all these solutions together. In this study, we make progress toward addressing this shortcoming of the field.

The pattern formation problem has also been studied from an operations research perspective. A solution has been proposed that formulates the problem as a second-order cone program [7] and uses interior-point methods to solve it. This solution, however, relies on a prescribed assignment and does not consider reflection. The authors report a constant runtime, but this is in the number of iterations of the convex optimization step, and does not consider the time to create the necessary data structures. Our implementation has a time-complexity of $O(n^3)$ where n is the number of robots in the system. Some work has been done to incorporate assignment as well, but current solutions exist only for minimizing the *total* distance traveled by all robots (as opposed to the maximum distance traveled by any robot in the system) [1]. While these solutions are practical and useful for many situations, they are not analytical and do not provide any insight into the properties of optimal solutions. In this study, we develop a theoretical understanding of the pattern formation problem and work toward an analytical solution to the problem.

4 Necessary Conditions

First, we start by characterizing an optimal solution. In this section we present the necessary conditions that every optimal solution must satisfy.

Critical Robots. Throughout the paper, we use *critical robots* to refer to robots which move the maximum distance (the solution). In Lemma 1 we show that

in any optimal solution there are at least three critical robots.

Lemma 1 *Given a system of n robots with initial positions $R = (r_0, r_1, \ldots, r_{n-1})$, let d^* be the optimal solution for forming some pattern. Then at least three robots traverse exactly distance d^*.*

Lemma 1 does not prove the existence of any upper bound on the number of robots that move distance d^*.

5 Replication

In this section we present *Replication* as a tool that we use to derive results presented later in the study. The replication machine is based on pure geometry and resembles a Pantograph. While replication is naturally applicable for any pattern with three or more vertices, we present replication for triangles in this study to simplify notation and proofs.

Definition 1 (Trivial Replication) *The Trivial Replication of a triangle P on a pair of points (u, v) is the triangle rigidly similar to P whose first two points are fixed to u and v. Formally:*

$$R_{Triv}(P, u, v) = T \overset{*}{\sim} P \text{ such that } t_0 = u, \text{ and } t_1 = v.$$

For any Trivial Replication $T = R_{Triv}(P, u, v)$, we call $u = t_0$ and $v = t_1$ its anchors. We call the third point, t_2, the *Trivial Replication Point*. Note that the Trivial Replication Point is not explicitly fixed to a prescribed point, rather, its position is entirely dependent on the triangle being replicated and the two anchors.

Definition 2 (Replication Machine) *The Replication Machine of a triangle P on a point and a circle $(u, C(v, r))$ is the infinite set of triangles rigidly similar to P whose first point is fixed to u and whose second point is on the circle $C(v, r)$. Formally:*

$$R_{Mach}(P, u, v, r) = \{T \overset{*}{\sim} P | t_0 = u, t_1 \in C(v, r)\}.$$

or equivalently:

$$R_{Mach}(P, u, v, r) = \{R_{Triv}(P, u, v') | v' \in C(v, r)\}.$$

Observe that $R_{Mach}(P, u, v, r)$ is the set of all patterns rigidly similar to P such that, for any $T \in R_{Mach}(P, u, v, r)$, t_0 is fixed to u and t_1 is exactly distance r from v. Observe that each triangle in a Replication Machine is also a Trivial Replication of the same triangle. We call the set of Trivial Replication Points of the Trivial Replications in a Replication Machine *Replication Machine Points*.

Definition 3 (Replication Spanner) *The Replication Spanner of a triangle P on a pair of circles $(C(u, r), C(v, r))$ is the infinite set of triangles rigidly similar to P whose first and second points are on the circles $C(u, r)$ and $C(v, r)$, respectively. Formally:*

$$R_{Span}(P, u, v, r) = \{T \overset{*}{\sim} P | t_0 \in C(u, r), t_1 \in C(v, r)\}.$$

or equivalently:

$$R_{Span}(P, u, v, r) = \bigcup_{u' \in C(u,r)} R_{Mach}(P, u', v, r).$$

$R_{Span}(P, u, v, r)$ is the set of all patterns rigidly similar to P such that, for any $T \in R_{Span}(P, u, v, r)$, both t_0 and t_1 are exactly distance r from p and q, respectively. We call the set of Trivial Replication Points of the Trivial Replications in a Replication Spanner *Replication Spanner Points*.

It is starting to become clear why Replication is a useful tool for pattern formation. Suppose u and v are the initial positions of two robots in a system that must form a triangle P. Then $R_{Span}(P, u, v, r)$ is the set of all patterns rigidly similar to T that the robots can form by each moving distance r. Since we are dealing with a system of three robots (forming triangular patterns), we know that all three robots are *critical* (Lemma 1). Therefore, the optimal pattern (without considering permutation or reflection) must be one from $R_{Span}(P, u, v, r)$ for some value of r.

Lemma 2 *Let c be the Trivial Replication Point of a triangle P on a pair of points (u, v). Then the set Replication Machine Points of P on $(u, C(v, r))$ is enclosed by the circle $C\left(c, r\frac{d(u,c)}{d(u,v)}\right)$.*

Proof. [Proof sketch] First, we show that the Replication Machine Points form a circle. Consider the Trivial Replication $T = R_{Triv}(P, u, v)$ (note that $t_2 = c$) and an arbitrary Trivial Replication $M \in R_{Mach}(P, u, v, r)$ (note that m_2 is in the Trivial Replication Circle of $R_{Mach}(P, u, v, r)$). First, observe that since T is rigidly similar to M, then $d(u, m_1) = k\ d(u, v)$ and $d(u, m_2) = k\ d(u, c)$ for some k, thus $\triangle u m_2 c$ is similar to $\triangle u m_1 v$ and $d(c, m_2)$ must be proportional to r.

In order to simplify the calculation of the circle's radius, consider the Trivial Replication $M \in R_{Mach}(P, u, v, r)$ such that m_1 is colinear with the line $\overleftrightarrow{uv}$. Observe that $k\ d(u, v) = d(u, v) + r$ and $k\ d(u, c) = d(u, c) + d(c, m_2)$. Solving the system of equations results in $d(c, m_2) = r\frac{d(u,c)}{d(u,v)}$. □

We call $C\left(c, r\frac{d(u,c)}{d(u,v)}\right)$ the Replication Machine Circle of $R_{Mach}(P, u, v, r)$.

Lemma 3 *If c is the Trivial Replication Point of a triangle P on a pair of points (u, v). Then $C\left(c, r\frac{d(u,c)+d(v,c)}{d(u,v)}\right)$ is the smallest circle that encloses the Replication Spanner Points of P on $(C(u, r), C(v, r))$.*

Proof. [Proof sketch] Consider the Trivial Replication $T = R_{Triv}(P, u, v)$, and the replication machines $\mathcal{M} = R_{Mach}((p_1, p_0, p_2), v, u, r)$ and, for some $M \in \mathcal{M}$, $\mathcal{S} = R_{Mach}(P, m_0, v, r)$.

Observe that for any $S \in \mathcal{S}$, by the definition of Replication Spanner, $S \in R_{Span}(P, u, v, r)$. Observe that the center of the Replication Machine Circle of $\mathcal{S}$ is in the Replication Machine Circle of $\mathcal{M}$. Therefore, c is the center-of-centers of two Replication Machine Circles. □

We call $C\left(c, r\frac{d(u,c)+d(v,c)}{d(u,v)}\right)$ the Replication Spanner Circle of $R_{Span}(P, u, v, r)$.

6 Three-Robot Solution

In this section, we present the main contribution of this study: a solution for systems of three robots. First, we show the optimal solution under rigid similarity, that is, we do not consider assignment (i.e. robot i with initial position r_i will assume the role of p_i in the desired pattern). Note that this is not necessarily the *optimal* solution. For systems of three robots, there are 3! possible assignments (permutations) of P that could be optimal. After presenting the solution for the trivial assignment, we demonstrate a simple method for choosing the correct assignment without testing all $3! = 6$ possibilities. Algorithm 1 produces a construction based entirely on geometric properties.

Algorithm 1 Algorithm for robot i in system with current positions R to form pattern P

// Let the perimeter of P be 1
// indices are modulo 3
1: $t_i \leftarrow$ point such that $\angle t_i r_{i+1} r_{i-1} = \angle p_i p_{i+1} p_{i-1}$ and $\angle r_{i+1} r_{i-1} t_i = \angle p_{i+1} p_{i-1} p_i$
2: $r \leftarrow d(r_i, t_i)\ d(p_{i+1}, p_{i-1})$
3: r_i moves r toward t_i

Lemma 4 *For any system of robots with initial positions R and any triangular pattern P, the distance r computed in Algorithm 1 (Line 2) is the same for each robot.*

Recall that for systems of three robots, all robots travel exactly the same distance. We show in Lemma 4 that Algorithm 1 satisfies this necessary condition.

Theorem 5 *For any system of robots with initial positions R and triangular pattern P with perimeter* 1*, let Q be the positions that robots move to after running Algorithm 1. Then Q is a valid solution. In other words, Q is similar to P.*

Algorithm 1 computes a valid solution such that all robots move the same distance. These conditions are necessary for any optimal solution, although not sufficient. We show in Theorem 6 that the solution Algorithm 1 produces is optimal.

Theorem 6 *For any system of robots with initial positions R and triangular pattern P, let Q be the positions that each robot moves to after running Algorithm 1, then Q is an optimal formation under rigid similarity.*

Optimal Pattern Formation by Three Robots. In order to prove Algorithm 1 is optimal, we assumed that robots move directly to their computed destinations. In Section 1, though, we discussed models where each robot executes *look*, *compute*, *move* cycles. In other words, we want to consider systems in which robots move a small distance ϵ toward their target, then re-compute the solution based on the new system state. In this section, we show that our solution is valid for models with oblivious robots.

Consider a modification of Algorithm 1, where instead of moving incrementally toward (rather than directly to) their destinations by replacing line 3 with:

$$r_i \leftarrow \text{ moves } min(r, \epsilon) \text{ toward } t_i$$

Theorem 7 *Let $f_i(t)$ denote the position of robot i at time t. For any $\epsilon > 0$, let Q^t be the solution computed at time t. Then, $Q^t = Q^{t+1}$.*

Assignment. The geometric construction provides a solution under rigid similarity only and therefore does not consider different assignments (permutations) of the desired pattern. In order to find the globally optimal solution, the geometric construction must be considered for all permutations of P. In this section, we present a simple method for choosing the optimal assignment without testing all $3! = 6$ possibilities.

Theorem 8 *Consider a system of robots with initial positions $R = (r_0, r_1, r_2)$, a pattern $P = (p_0, p_1, p_2)$, and $d(r_0, r_1) \leq d(r_1, r_2) \leq d(r_2, r_0)$. Then P is an optimal assignment for R if and only if $d(p_0, p_1) \leq d(p_1, p_2) \leq d(p_2, p_0)$.*

Observe that, for any triangle P, $d(p_0, p_1) \leq d(p_1, p_2) \leq d(p_2, p_0)$ if and only if $\angle p_{i-1} p_0 p_{i+1} \leq \angle p_0 p_{i+1} p_{i-1} \leq \angle p_{i+1} p_{i-1} p_0$. Theorem 8 indicates that the optimal formation can be obtained by first sorting R and P by their angles (or side lengths), and then running Algorithm 1.

Corollary 9 *For a system of robots with initial positions $R = (r_0, r_1, r_2)$, such that $d(r_0, r_1) \leq d(r_1, r_2) \leq d(r_2, r_0)$ and a pattern $P = (p_0, p_1, p_2)$ such that $d(p_0, p_1) \leq d(p_1, p_2) \leq d(p_2, p_0)$ let Q be the positions that robots move to after running Algorithm 1. Then Q is an optimal formation.*

Proof. Follows from Theorems 6 and 8. □

7 Triangle Metric

In this section, we introduce a metric for comparing triangles inspired by the solution for systems of three robots presented in Section 6. Let $d^*(A, B)$ be the optimal distance that robots with initial positions A need to form B. This distance can also be interpreted as a distance between the triangles A and B. d^* is not a valid metric by itself, though. In particular, since d^* depends on the position and size of the first argument only, it is not symmetric, or $d^*(A, B) \neq d^*(B, A)$. In order to enforce symmetry, our metric should be invariant to translation, rotation, uniform scaling, reflection, and permutation of both A and B.

Lemma 10 *Let α and β the ordered sequences of interior angles of two triangles. Then τ is a valid metric for comparing the triangles:*

$$\tau^2(\alpha, \beta) = \frac{sin^2(\alpha_1)}{sin^2(\alpha_2)} + \frac{sin^2(\beta_1)}{sin^2(\beta_2)} - 2\frac{sin(\alpha_1)\ sin(\beta_1)}{sin(\alpha_2)\ sin(\beta_2)}\ cos(\alpha_0 - \beta_0)$$

The τ-distance between two triangles, defined by their angles, is a measure of similarity between them. Two triangles, A and B are similar when $\tau(A, B) = 0$. If $\tau(A, B) < \tau(A, C)$ this indicates that B is *more similar* to A than C is. In other words, a system of robots with an initial formation of A would need to travel further to form C than it would move in order to form B.

8 Arising Geometric Properties

In this section, we present some interesting properties of systems of three robots and the patterns they can form.

Focal Point. One interesting property that emerges for every system of three robots forming any arbitrary pattern is that all three of their paths can be characterized by a single point on the plane.

Theorem 11 *For systems of three robots and any optimal formation, there exists a point that all robots move either directly toward or directly away from.*

For any optimal pattern, we call this single point that robots move either directly to or from, the *focal point*.

Constant Center-of-Mass. For systems forming equilateral triangles, an even stronger property emerges.

Lemma 12 *Suppose Q is an optimal formation for a system of three robots with initial positions R to form an equilateral triangle. Then, the center of mass of Q is equivalent to that of R. Furthermore, since robots move at the same speed, the system's center of mass is invariant with respect to time.*

9 Conclusion

The main contribution of this study is an optimal solution for systems of three robots. Systems of three robots are interesting because they have clear applications to systems of many robots. Recall that, even in the general case, at least three robots must traverse the maximum distance, therefore it is a lower bound for the general case, that is, d is the minimum optimal solution for all combinations of three robots and triangular sub patterns of P with a prescribed assignment, or:

$$d = \min_{p_i,p_j,p_k \in P} \left(\max_{r_i,r_j,r_k \in R} (d^*((r_i, r_j, r_k), (p_i, p_j, p_k))) \right)$$

Finding an upper bound on the solution is an area for future work. A generalized Replication Machine tool might prove useful in finding the solution for systems of n robots.

We are also interested in finding an algorithm for determining the optimal assignment in the general case. It is clear that some assignments are infeasible. For example, it makes intuitive sense that a robot's set of nearest neighbors in the initial configuration of the system should be close to that of final configuration.

Further work is also needed to understand under which models (see Section 1) our solution (or some variant of it) is valid for. For example, Algorithm 1 is only valid for synchronous models, where each robot starts its cycle at the same time (according to a global clock). If the robots were asynchronous, they would compute optimal solutions for different initial configurations, since they would observe the current positions of robots at different times.

Finally, we plan to explore applications for the triangle metric introduced in Section 7. The metric provides a nice way to score, classify, or sort triangles based on their similarity to each other. This has potential applications in computer vision, computational geometry, and of course, mobile robotics.

Acknowledgements. This work was initiated at the 18th Routing Workshop which took place in Merida, Mexico from July 29 to August 02, 2019. Research supported by PAPIIT grant IN 102117 from Universidad Nacional Autónoma de México. B.V. was partially supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922 and by the Austrian Science Fund within the collaborative DACH project *Arrangements and Drawings* as FWF project I 3340-N35.

References

[1] S. Agarwal and S. Akella. Simultaneous optimization of assignments and goal formations for multiple robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6708–6715, 2018.

[2] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart, and P. Beardsley. Multi-robot system for artistic pattern formation. In *2011 IEEE International Conference on Robotics and Automation*, pages 4512–4517, 2011.

[3] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

[4] R. Cohen and D. Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34(6):1516–1528, 2005.

[5] R. Cohen and D. Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science*, 399(1-2):71–82, 2008.

[6] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. On the computational power of oblivious robots: forming a series of geometric patterns. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 267–276, 2010.

[7] J. Derenick and J. Spletzer. Convex optimization strategies for coordinating large-scale robot formations. *IEEE Transactions on Robotics*, 23(6):1252–1259, 2007.

[8] P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Distributed computing by mobile robots: Solving the uniform circle formation problem. In *International Conference on Principles of Distributed Systems*, pages 217–232, 2014.

[9] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005.

[10] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1):412–447, 2008.

[11] R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. *Theoretical Computer Science*, 390(1):27, 2008.

[12] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[13] H. Oh, A. R. Shirazi, C. Sun, and Y. Jin. Bio-inspired self-organising multi-robot pattern formation. *Robotics and Autonomous Systems*, 91:83–100, 2017.

[14] T. Schetter, M. Campbell, and D. Surka. Multiple agent-based autonomy for satellite constellations. *Artificial Intelligence*, 145(1-2):147–180, 2003.

[15] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

Path Planning in a Weighted Planar Subdivision Under the Manhattan Metric

Mansoor Davoodi* † Hosein Enamzadeh* ‡ Ashkan Safari* §

Abstract

In this paper, we consider the problem of path planning in a weighted polygonal planar subdivision. Each polygon has an associated positive weight which shows the cost of path per unit distance of movement in that polygon. The goal is finding a minimum cost path under the Manhattan metric for two given start and destination points. We propose an $O(n^2)$ time and space algorithm to solve this problem, where n is the total number of vertices in the subdivision. We also study the case of rectilinear regions in three dimensions, and generalize the proposed algorithm to find a minimum cost path under the Manhattan metric in $O(n^3 \log n)$ time and $O(n^3)$ space.

1 Introduction

Path planning (PP) problem is one of the fundamental problems in motion planning whose objective is to find an optimal path with minimum length between two start and destination points s and t in a work space. In the classical version of PP, the work space contains some obstacles, and the path must avoid these obstacles [7, 14]. However, in a general formulation of PP – called *Weighted Region Problem* (WRP) – which was first introduced by Mitchell and Papadimitriou [17], each obstacle has an associated weight and a path is allowed to enter them at extra costs. In fact, these weights represent the cost per unit distance of movement in the obstacles (or say *weighted regions*). This generalization of PP has a lot of applications, e.g., it can be used in self-driving cars navigation, robot motion planning [6], military purposes [16], crowd simulation [13], and gaming applications [13]. An important theoretical result on WRP [9] has shown that this problem cannot be solved in the algebraic computation model over the rational numbers under the Euclidean metric. Motivated by this result, we investigate WRP under the Manhattan metric and show that it can be solved efficiently in polynomial time.

*Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran

†mdmonfared@iasbs.ac.ir

‡hosein.enamzadeh@iasbs.ac.ir

§ashkan.safari@iasbs.ac.ir

Mitchell and Papadimitriou [17] introduced an ϵ-optimal algorithm with running time of $O(n^8 L)$, where n is the total number of vertices of polygonal regions and L is the precision of problem's instance. Precisely, $L = O(\log(nNW/\epsilon w))$, where N is the maximum integer coordinate of any vertex of the subdivision, W and w are the maximum non-infinite and minimum non-zero integer weights assigned to the faces of the subdivision, and $\epsilon > 0$ is a user-specified error tolerance. The output is the shortest path from the starting point s to all vertices of the polygons with an error tolerance ϵ under the Euclidean metric. Mata and Mitchell [16] have proposed an algorithm based on constructing a relatively sparse graph – called *pathnet* – that can search for paths that are close to optimal. They have proved that a pathnet of size $O(nk)$ can be constructed in $O(kn^3)$ time. As a matter of fact, the pathnet limits the paths that can extend from vertices with k cones at each vertex. Searching for a path on the constructed pathnet yields a path whose weighted length is at most $(1 + \epsilon)$ of optimal path. Precisely, $\epsilon = \frac{W/w}{k\Theta_{min}}$, where W/w is the ratio of the maximum non-infinite weight to the minimum non-zero weight, and θ_{min} is the minimum internal face angle of the subdivision. One of the common techniques for obtaining approximate shortest paths is to positioning Steiner points for discretizing the edges of the triangular regions and then constructing a graph by connecting them. Finally, by using graph search algorithms such as Dijkstra, an approximate minimum cost path can be computed [1, 2, 18].

There are several variants of WRP due to the metric and the shape of weighted regions. Lee et al. [15] have solved the problem in the presence of isothetic obstacles (the boundary edges of obstacles are either vertical or horizontal line segments). They have presented two algorithms for finding the shortest path under the Manhattan metric. The first algorithm runs in $O(n \log^2 n)$ time and $O(n \log n)$ space, and the second one runs in $O(n \log^{3/2} n)$ time and space. Gewali et al. [10] have considered a special case of this problem in which there are only three types of regions: regions with weight of ∞, regions with weight of 0, and regions with weight of 1. They have presented an algorithm in $O(m + n \log n)$ time, where $m \in O(n^2)$ is the number of visibility edges. Furthermore, they have presented an algorithm for the case that linear feathers are added. Precisely, edges of

the subdivision are allowed to have arbitrary weights. Their algorithm for this case takes $O(n^2)$ time for constructing a graph of size $O(n^2)$ for searching the shortest path. In fact, it takes $O(n^2 \log n)$ time for finding the shortest path. Gheibi et al. [11] have discussed the problem in an arrangement of lines. Due to the fact that this special case of the problem has unbounded regions, they have presented a minimal region – called *SP-Hull* – to bound the regions. This minimal region contains the minimum cost path from s to t. They construct *SP-Hull* in $O(n \log n)$ time, where n is the number of lines in the arrangement. After constructing *SP-Hull*, an approximate minimum cost path can be obtained by applying the existing approximation algorithms within bounded regions. Jaklin et al. [13] have analyzed the problem when the weighted regions are cells of a grid. They have also presented a new hybrid method – called *vertex-based pruning* – which is able to compute paths that are ϵ-optimal inside a pruned subset of the scene.

In this paper, we consider a planar subdivision with arbitrary positive weights. We present an algorithm which constructs a planar graph in $O(n^2)$ time with $O(n^2)$ vertices and edges, where n is the total number of vertices of the subdivision. The constructed graph contains the minimum cost path between two points s and t in the plane, where the distances are measured under the weighted Manhattan metric – the length of a path is the weighted sum of Manhattan lengths of the sub-paths within each region. It has been shown that this problem is unsolvable over the rational numbers when the distances are measured under the weighted Euclidean metric [9]. To the best of our knowledge, this is the first result that presents an exact algorithm for solving WRP under the Manhattan metric in a case where the regions are arbitrary simple polygons with positive weights. We propose an exact algorithm for finding the minimum cost path under the weighted Manhattan metric in $O(n^2)$ time which is also a $\sqrt{2}-$approximation for the Euclidean metric. Also, we show that the proposed algorithm can be used for WRP with rectilinear subdivision in three dimensions in $O(n^3 \log n)$ time and $O(n^3)$ space.

This paper is organized in five sections. In section 2, we give some preliminaries and definitions. In section 3, we present our algorithm for constructing a graph which contains the minimum cost path in a two dimensional work space, and prove that the shortest path is within the constructed graph. In section 4, we generalize the algorithm for the case of rectilinear regions in three dimensions, and in section 5, we draw a conclusion.

2 Preliminaries and Definitions

The problem of weighted region path planning, WRP, considered in this paper is defined as follows: let $\mathcal{S}$ be a subdivision of the plane into polygonal regions with n vertices, and $s, t \in \mathcal{S}$ be two start and destination points in the plane. Each region of $\mathcal{S}$ has an associated positive weight. The weight of an edge $e \in \mathcal{S}$ (boundary of regions) is assumed to be $\min\{w_r, w_{r'}\}$, where w_r and $w_{r'}$ are the weights of regions incident to e. The goal is to find a minimum cost path between s and t, where the distances are measured under the weighted Manhattan metric – the length of a path is the weighted sum of Manhattan lengths of the sub-paths within each region.

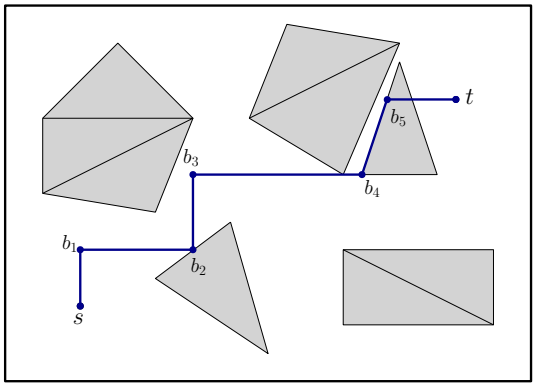


Figure 1: A path from s to t with seven breakpoints.

Let π_{st} denote a path between s and t which consists of some sub-paths between consecutive *breakpoints*. A breakpoint is a point on the path in which the path turns. We also consider s and t as breakpoints (see Fig. 1). Let $\rho_1, \rho_2, ..., \rho_k$ be sub-paths between consecutive breakpoints of a path π_{st} in which each ρ_i, for $i = 1, 2, \ldots, k$ lies completely within one region. If a part of a path π_{st} does not lie totally in one of the regions, we decompose it to some sub-paths. We denote $d(\rho_i)$ as the Manhattan distance between two endpoints of ρ_i. The weighted length of a path π_{st} under the Manhattan metric, denoted by $d_w(\pi_{st})$, is defined as:

$$d_w(\pi_{st}) = \sum_{i=1}^{k} d(\rho_i) \times w_i,$$

where w_i is the weight of the region in which ρ_i lies.

A path π_{st} is called a horizontal (resp., vertical) path if it consists of a horizontal (resp., vertical) sub-path between only two consecutive breakpoints. Also, we say two horizontal (resp., vertical) paths are consecutive if and only if they have the same starting and termination points. This definition is used in Lemma 1.

The basic idea behind the proposed algorithm is reducing the problem to a graph searching problem. Therefore, we provide an algorithm for constructing a graph that contains the minimum cost path under the weighted Manhattan metric. The constructed graph is a planar graph with $O(n^2)$ vertices and edges, where n is the total number of vertices of the subdivision. For planar graphs with positive edge weights, Henzinger et al. [12] have given a linear-time algorithm to compute

single-source shortest paths. By running this algorithm on the constructed graph, we obtain the minimum cost path between s and t under the Manhattan metric in $O(n^2)$ time. Since a simple polygon with n vertices can be triangulated in $O(n \log n)$ time and $O(n)$ space [8], w.l.o.g. we assume all the regions to be triangular regions in all parts of the paper.

3 The Graph Construction Algorithm

3.1 The Algorithm

Let $\mathcal{G} = (V, E)$ be a graph. First, we initialize $V = \emptyset$ and $E = \emptyset$. Let $HL(\alpha_i)$ and $VL(\alpha_i)$ be horizontal and vertical lines passing through point α_i, for $i = 1, 2, \ldots, n$. Precisely, α_i, for $i = 1, 2, \ldots, n$ are the vertices of the subdivision which contain s, t, and the vertices of the triangles. We add s, t, vertices of the triangles, and the intersection points among $HL(\alpha_i)$ and $VL(\alpha_j)$, for $i, j = 1, 2, \ldots, n$ to V. We also add the intersection points among $HL(\alpha_i)$ (resp., $VL(\alpha_i)$), for $i = 1, 2, \ldots, n$ and the edges of the triangles to V. Next, we add the line segments between two consecutive vertices in V that lie on the considered horizontal lines, vertical lines or the edges of the triangles as edges of $\mathcal{G}$ to E. For an edge $(u, v) \in E$ where lies in a region with wight w_i, let $d(u, v)$ denote the Manhattan distance between two endpoints of the edge. The weight of the edge is equal to the product of $d(u, v)$ and w_i. Note that each edge lies completely within one region.

The basic idea of our algorithm is to extend four rays to the up, down, right and left directions (horizontal and vertical lines) at every vertex of the subdivision. This idea has similarity to vertical cell decomposition (VCD) method [14]. In this method, the free space is partitioned into a finite collection of one-dimensional and two-dimensional cells by extending rays upward and downward through free space. In this method, the rays are not allowed to enter obstacles, however, in our algorithm the rays are extended to all parts of the subdivision since the paths are allowed to enter weighted regions at extra costs. Also, we extend rays to the four directions at every vertex, however, in the VCD method the rays are extended only upward and downward. In both methods, the motion planning problem is reduced to a graph search problem. In VCD method, a roadmap is constructed by selecting sample points from the cell centroids, however, in our algorithm the graph is constructed by intersecting the rays with each other and also by the edges of the triangles.

Some of the edges of $\mathcal{G}$ which lie on an edge of a triangle are oblique. These edges are useful when two triangular regions are close to each other and the region among them has a lower weight than these triangles. A path which passes between these two triangles cannot be completely horizontal or vertical since it will enter the triangles. So it will be oblique and lie on one of the edges of the triangles (see the sub-path between b_4 and b_5 on Fig. 1).

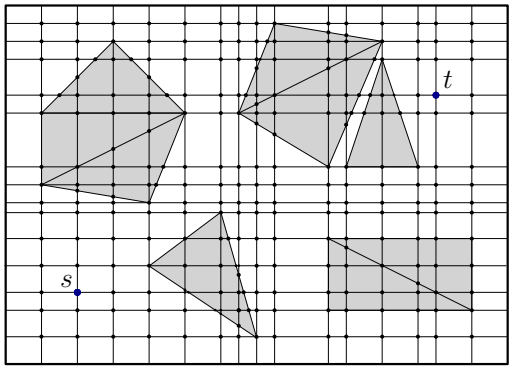


Figure 2: The constructed graph of Fig. 1.

According to the construction of the graph, some vertices and edges are added to the graph by vertical and horizontal lines passing through vertices of the subdivision. We call the part of the work space which lies between two consecutive horizontal (resp., vertical) lines, a *horizontal lane* (resp., *vertical lane*) denoted by LH (resp., LV). So each LH (resp., LV) is surrounded by two consecutive horizontal (resp., vertical) lines. Therefore, when we say the lines of an LH (resp., an LV), we mean these consecutive lines.

For constructing the graph, we can use one of the line segments intersections algorithms [3, 5] which computes all k intersections among n line segments in the plane in $O(n \log n + k)$ time. These intersection points are vertices of $\mathcal{G}$. After specifying the set of vertices of $\mathcal{G}$, the set of edges of $\mathcal{G}$ can be specified. It takes $O(n^2)$ time to construct $\mathcal{G}$ since the graph has $O(n^2)$ vertices and edges. The constructed graph of the work space on Fig. 1 is shown on Fig. 2. For simplicity, we do not triangulate the white regions with weight 1 in these figures. Precisely, we can apply the proposed algorithm in a polygonal subdivision in which the regions are not triangular. The triangulation of the regions just helps us for showing that $\mathcal{G}$ contains the minimum cost path between s and t.

For computing the minimum cost path under the Manhattan metric between s and t, we can apply Dijkstra's algorithm to $\mathcal{G}$. In this case, the minimum cost path is obtained in $O(n^2 \log n)$ time. However, since $\mathcal{G}$ is a planar graph with positive edge weights, we can apply the algorithm presented by Henzinger et al. [12], which is a linear-time algorithm, to $\mathcal{G}$. Therefore, the minimum cost path is obtained in $O(n^2)$ time.

3.2 Correctness Proof

Now, we show that the constructed graph contains the minimum cost path between s and t under the Manhat-

tan metric. Since our metric for measuring the distance is Manhattan, we can convert any path between s and t to a path which consists of vertical and horizontal line segments. In other words, when a sub-path between two consecutive breakpoints is oblique, we can replace it by two horizontal and vertical line segments where the cost of movement on these horizontal and vertical line segments is equal to the cost of movement along the oblique line segment. In a case where a sub-path lies between two close triangular regions and the region between these two triangular regions has lower weight than these triangles, by applying this conversion, some parts of the horizontal and vertical line segments may lie in the triangular region with higher weight. In this case, we can replace the part which lies in a triangular region with higher cost with a line segment which lies on an edge of the triangles (see the sub-path between b_4 and b_5 on Fig. 1). Since the weight of each of the edges of the work space is equal to the minimum weight of the regions that are incident to that edge, the cost of movement between two breakpoints on the replaced line segments is equal to the cost of movement along the oblique line segment. Therefore, a path between s and t can only consist of horizontal, vertical, and oblique line segments, the latter of which are located on the edges of the triangles. As a result, all the paths that we consider in the following lemmas consist of the above mentioned line segments. Our first objective is to prove the following lemma.

Lemma 1 *Let π_1, π_2, and π_3 be three consecutive horizontal (or vertical) sub-paths from s' to t' which lie inside an LH (resp., an LV) and pass through $k > 0$ triangular regions. If $d_w(\pi_2) < d_w(\pi_1)$, then $d_w(\pi_3) < d_w(\pi_2)$.*

Proof. We consider the case $k = 2$, the proof is similar for any $k > 0$. For simple comparison among the sub-paths, let the points s' and t' lie on the same horizontal line segment. Assume w.l.o.g. that both triangles have vertical edges (see Fig. 3). The weighted lengths of π_1, π_2 and π_3 are defined as follows (refer to Fig. 3 for the notations):

$$d_w(\pi_1) = (w_1 \times a_1) + (w_2 \times a_2) + z_2 + x_2 + L,$$

$$d_w(\pi_2) = (2 \times h) + x_1 + (w_1 \times b_1) + (w_2 \times b_2) + x_2 + L,$$

$$d_w(\pi_3) = (2 \times h) + x_1 + (2 \times h') + z_1 + (w_1 \times c_1) + (w_2 \times c_2) + L.$$

According to Fig. 3, $a_1 = b_1 + x_1$ and $a_2 = b_2 - z_2$. Due to the assumption that $d_w(\pi_2) < d_w(\pi_1)$, we have the following inequality:

$$(2 \times h) < x_1 \times (w_1 - 1) + z_2 \times (1 - w_2),$$

and due to the triangle similarity theorems we have the following equations:

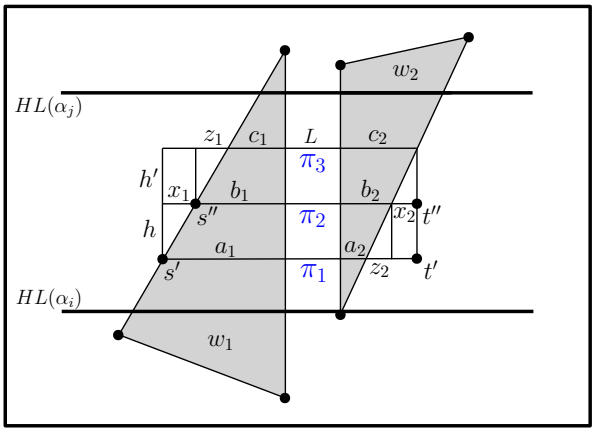


Figure 3: Three consecutive horizontal sub-paths from s' to t' through two triangular regions.

$$\frac{x_1}{h} = \frac{z_1}{h'}, \frac{z_2}{h} = \frac{x_2}{h'}.$$

By applying the triangle similarity equations in the mentioned inequality and adding $(w_1 \times c_1) + (w_2 \times b_2)$ to both sides of the inequality we get:

$$(2 \times h') + z_1 + (w_1 \times c_1) + (w_2 \times c_2) < (w_1 \times b_1) + (w_2 \times b_2) + x_2 \Longrightarrow d_w(\pi_3) < d_w(\pi_2).$$

Thus, the weighted length of π_3 is less than π_2. In fact, the proof is based on the following equation:

$$\frac{h}{h'} = \frac{x_1}{z_1} = \frac{z_2}{x_2},$$

and since $\frac{h}{h'}$ is constant, we can generalize the proof for any $k > 0$ triangular regions between s' and t'. Therefore, the lemma holds. □

Note that inside an *LH* (resp., an *LV*), we can consider all the triangles to have vertical (resp., horizontal) edges since vertical (resp., horizontal) lines are considered passing through vertices of the subdivision. The result of this lemma helps us to show that there exists a shortest path between s and t under the Manhattan metric such that all the horizontal (resp., vertical) sub-paths between consecutive breakpoints in *LHs* (resp., *LVs*) lie on the lines of the *LHs* (resp., *LVs*). We call such a path, a *perfect shortest path* between s and t, denoted by π^p_{st}. Note that according to the principle of optimality, since π^p_{st} is optimal in length, all of its sub-paths in *LHs* and *LVs* are also optimal in length.

Lemma 2 *There exists a shortest path between s and t under the Manhattan metric such that, for any sub-path of the shortest path in an LH (resp., an LV), all the horizontal (resp., vertical) sub-paths between consecutive breakpoints lie on the lines of the LH (resp., LV).*

According to Lemma 2, a path between the entrance (s') and exit point (t') of an LH (resp., an LV) is not optimal in length, unless there exists an optimal path in length such that all the horizontal (resp., vertical) sub-paths between consecutive breakpoints lie on the lines of the LH (resp., LV). Precisely, there is always a path $\pi^p_{s't'}$ in an LH (resp., an LV). According to the construction of the graph, lines of an LH (resp., an LV) are edges of $\mathcal{G}$ and a horizontal (resp., vertical) sub-path of a path $\pi^p_{s't'}$ between two consecutive breakpoints in an LH (resp., an LV) lies on the edges of $\mathcal{G}$.

Corollary 3 *For any path $\pi^p_{s't'}$ in an LH (resp., an LV), the sub-paths between consecutive breakpoints cannot be simultaneously horizontal (resp., vertical) and lie between two lines of the LH (resp., LV).*

Lemma 4 *A breakpoint of a path $\pi^p_{s't'}$ in an LH (resp., an LV) is located on a line of an LH or an LV or possibly both.*

Proof. We assume that b is a breakpoint in an LH which is not located on a line of the LH or a LV. According to Corollary 3, the line segment that is incident to b cannot be horizontal. Therefore, one of the line segments is vertical and the other one is located on an edge of a triangle. Since b is also located in an LV and is not located on one of the lines of the LV, the vertical line segment incident to b lies between the left and right lines of the LV, which contradicts Corollary 3. Thus, the lemma holds. □

Lemma 4 shows that the breakpoints of the perfect shortest paths in *LHs* (resp., *LVs*) must lie on the lines of the *LHs* and *LVs*, meaning that they lie on the edges of $\mathcal{G}$ (since the lines of *LHs* and *LVs* are edges of G). The next step is to show that these breakpoints are located on the vertices of $\mathcal{G}$.

Lemma 5 *For a path $\pi^p_{s't'}$ in an LH (resp., an LV), the breakpoints of the path are located on the vertices of $\mathcal{G}$.*

Proof. According to Lemma 4, a breakpoint of a path $\pi^p_{s't'}$ in an LH (resp., an LV) is located on a line of an LH or an LV or possibly both. If a breakpoint is located on both a line of an LV and a line of an LH, it is on the intersection point of these two lines. Thus, it is on a vertex of $\mathcal{G}$. If it is only located on a line of an LH or an LV, and one of the incident line segments lies on a triangle edge, then the breakpoint is located on a vertex of $\mathcal{G}$ (since the intersection of an LH or LV line with a triangle edge is a vertex of $\mathcal{G}$). Therefore, the breakpoints of a path $\pi^p_{s't'}$ are on the vertices of $\mathcal{G}$. □

Lemma 5 shows that the breakpoints of a path $\pi^p_{s't'}$ in an LH (resp., an LV) are located on the vertices of $\mathcal{G}$. The next step is to show that a path $\pi^p_{s't'}$ under the Manhattan metric in an LH (resp., an LV) is on $\mathcal{G}$. To this end, we need to show that the edges of the path $\pi^p_{s't'}$ are on the edges of $\mathcal{G}$.

Lemma 6 *A path $\pi^p_{s't'}$ in an LH (resp., an LV) is on $\mathcal{G}$.*

Proof. According to Lemma 5, the breakpoints of a path $\pi^p_{s't'}$ in an LH (resp., an LV) are on the vertices of $\mathcal{G}$. Let e be an edge between two consecutive breakpoints. If e is on an edge of a triangle, it is on $\mathcal{G}$. Now we assume that e is in an LH and is not on $\mathcal{G}$. According to Corollary 3, e cannot be horizontal since it must lie on one of the lines of the LH and the lines of LHs are edges of $\mathcal{G}$. Therefore, it is a vertical edge. Since it is also located in an LV and is not on $\mathcal{G}$, it is not on a line of the LV. Therefore, it contradicts Corollary 3. Thus, e is on $\mathcal{G}$. □

According to Lemma 6, perfect shortest paths in *LHs* and *LVs* which are sub-paths of a path π^p_{st} are on the constructed graph. Note that in all the lemmas, a path between s and t only consists of horizontal, vertical, and oblique line segments, the latter of which are located on the edges of the triangles. In the continuous work space, an arbitrary path between s and t consists of line segments which are not in the form of the mentioned line segments. Finally, we prove that there exists a shortest path between s and t on $\mathcal{G}$.

Theorem 7 *For a shortest path π_1 under the weighted Manhattan metric in the continuous work space from s to t, there exists a path π_2 from s to t on $\mathcal{G}$ such that $d_w(\pi_2) \leq d_w(\pi_1)$.*

Proof. It is obvious that when the metric for measuring the distance is Manhattan, any arbitrary path in the continuous work space, can be converted to a path which consists of the three mentioned line segments without increment in the cost of the path. Thus, we convert π_1 to π'_1 such that the line segments in π'_1 are in the form of the mentioned line segments. Obviously, $d_w(\pi'_1) = d_w(\pi_1)$. According to the principle of optimality, each sub-path of an optimal path in length is also optimal. Therefore, π'_1 consists of optimal sub-paths in length in *LHs* and *LVs*. According to Lemma 2, for any shortest path in an LH (resp., an LV), there exists a path $\pi^p_{s't'}$ and due to the Lemma 6, perfect shortest paths in *LHs* and *LVs* are on $\mathcal{G}$. Thus, π'_1 can be converted to a perfect shortest path (π_2) without increment in the cost of the path. Therefore, a path from s to t on $\mathcal{G}$ exists (π_2) whose weighted length is not greater than π_1. □

According to Theorem 7, $\mathcal{G}$ contains a shortest path from s to t under the weighted Manhattan metric. Since simple polygons can be triangulated in $O(n \log n)$ time

and $O(n)$ space [8], work spaces with simple polygonal regions can be discretized by using the mentioned graph construction algorithm. Thus, the proposed algorithm solves WRP under the Manhattan metric.

Theorem 8 *The weighted region problem in a planar polygonal subdivision with positive weights under the Manhattan metric can be solved in $O(n^2)$ time and space, where n is the total number of vertices of the subdivision.*

By using the triangular inequality, it is easy to see that the length of a path under the Manhattan metric is at most $\sqrt{2}$ times of the length of the path under the Euclidean metric. Thus, the proposed algorithm is also a $\sqrt{2}$-approximation algorithm for solving WRP under the Euclidean metric.

4 The Three-Dimensional Case

In this section, we consider WRP in three dimensions. It has been shown that the problem of finding a shortest path under any L^P metric in a three-dimensional polyhedral environment is NP-hard [4]. Here, we consider a specific variation where the regions are rectilinear.

Since the metric for measuring the distance is Manhattan, any oblique path between two consecutive breakpoints in three-dimensional space can be converted to three parallel line segments to x, y and z axes without increment in the cost of the path. Thus, we consider all the paths to be rectilinear.

Let n be the total number of vertices of the subdivision and let (x_i, y_i, z_i), for $i = 1, 2, \ldots, n$ be the coordinates of the vertices of the regions (and of s and t). Let $\mathcal{P}$ be the set of planes $x = x_i$, $y = y_i$, $z = z_i$, for $i = 1, 2, \ldots, n$. The set of vertices of the graph consists of the intersection points among at least three planes in $\mathcal{P}$, and the set of edges of the graph consists of the line segments between two consecutive vertices of the graph which lie on the intersection lines between at least two planes in $\mathcal{P}$. The constructed graph has $O(n^3)$ vertices and edges, and by applying Dijkstra's algorithm to it, the minimum cost path under the Manhattan metric can be obtained in $O(n^3 \log n)$ time.

Similar to the definitions of LH and LV in the planar case, we define similar notations for the three-dimensional case. Let XYC denote a part of the work space which is surrounded by two consecutive planes orthogonal to the x-axis and two consecutive planes orthogonal to the y-axis in $\mathcal{P}$ which is called an $XY-container$. Precisely, an XYC is not surrounded along the z-axis. XZC and YZC notations are defined similarly. Since all the paths are considered to be rectilinear, for any path in an XYC, there exists an equivalent path in length such that all the sub-paths between consecutive breakpoints along the z-axis are located on the planes surrounding XYC. Precisely, according to the graph construction algorithm, each XYC consists of some cuboids where the cost of movement in every part of a cuboid is equal. Therefore, the sub-paths along the z-axis in a cuboid have the same cost when they are located either on the planes surrounding XYC or inside the cuboid. Similar results hold for an XZC and a YZC. Thus, an equivalent path in length between s and t exists where all the sub-paths between consecutive breakpoints are located on the considered planes in $\mathcal{P}$. Arguments similar to the ones used in Theorem 7 show that the constructed graph contains the minimum cost path between s and t under the Manhattan metric.

Theorem 9 *The weighted region problem in a three-dimensional work space among rectilinear regions with positive weights under the Manhattan metric can be solved in $O(n^3 \log n)$ time and $O(n^3)$ space, where n is the total number of vertices of the subdivision.*

5 Conclusion

In this paper, we have considered a generalization of path planning problem – called *weighted region problem* (WRP). While unsolvability of WRP over the rational numbers under the Euclidean metric has been proved [9], we proposed an algorithm for solving WRP under the Manhattan metric which is also a $\sqrt{2}$-approximation solution for the Euclidean case. We also considered the case of rectilinear regions in three dimensions, and generalized our algorithm for it. Improving the time complexity of the algorithm and providing a better approximation factor for the Euclidean metric remain open.

References

[1] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ε—approximation algorithm for weighted shortest paths on polyhedral surfaces. In *Scandinavian Workshop on Algorithm Theory*, pages 11–22. Springer, 1998.

[2] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM (JACM)*, 52(1):25–53, 2005.

[3] I. J. Balaban. An optimal algorithm for finding segments intersections. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 211–219, 1995.

[4] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 49–60. IEEE, 1987.

[5] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM (JACM)*, 39(1):1–54, 1992.

[6] J. Chestnutt, K. Nishiwaki, J. Kuffner, and S. Kagami. An adaptive action model for legged navigation planning. In *2007 7th IEEE-RAS International Conference on Humanoid Robots*, pages 196–202. IEEE, 2007.

[7] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. C. Schwarzkopf. Polygon triangulation. In *Computational Geometry*, pages 45–61. Springer, 2000.

[9] J.-L. De Carufel, C. Grimm, A. Maheshwari, M. Owen, and M. Smid. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry*, 47(7):724–727, 2014.

[10] L. P. Gewali, A. C. Meng, J. S. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *ORSA Journal on Computing*, 2(3):253–272, 1990.

[11] A. Gheibi, A. Maheshwari, and J.-R. Sack. Weighted region problem in arrangement of lines. In *CCCG*, 2013.

[12] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *journal of computer and system sciences*, 55(1):3–23, 1997.

[13] N. Jaklin, M. Tibboel, and R. Geraerts. Computing high-quality paths in weighted regions. In *Proceedings of the Seventh International Conference on Motion in Games*, pages 77–86, 2014.

[14] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[15] D. Lee, C.-D. Yang, and T. Chen. Shortest rectilinear paths among weighted obstacle. *International Journal of Computational Geometry & Applications*, 1(02):109–124, 1991.

[16] C. S. Mata and J. S. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 264–273, 1997.

[17] J. S. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM (JACM)*, 38(1):18–73, 1991.

[18] Z. Sun and J. H. Reif. On finding approximate optimal paths in weighted regions. *Journal of Algorithms*, 58(1):1–32, 2006.

Appendix

Proof of Lemma 2

Lemma 2 *There exists a shortest path between s and t under the Manhattan metric such that, for any sub-path of the shortest path in an LH (resp., an LV), all the horizontal (resp., vertical) sub-paths between consecutive breakpoints lie on the lines of the LH (resp., LV).*

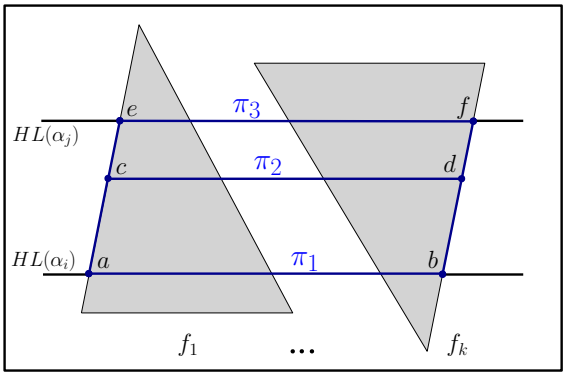


Figure 4: Three horizontal paths passing through k triangular regions.

Proof. Suppose the lemma for the case of a horizontal lane. Similarly, the lemma holds for a vertical lane. We consider s' as the entrance point to the LH and t' as the exit point. W.l.o.g. we consider that s' is on the left side of t'. Due to the assumption that the path between s and t is optimal in length, any sub-path of this path is also optimal in length. Thus, the path between s' and t' is optimal in length. We consider a path between s' and t' where a horizontal sub-path between two consecutive breakpoints does not lie on the lines of the LH. We show that there exists an equivalent path in length between s' and t' such that all the horizontal sub-paths between consecutive breakpoints lie on the lines of the LH. We assume c and d as two consecutive breakpoints such that the horizontal sub-path between them does not lie on the lines of the LH (see Fig. 4). There are k triangular regions between c and d and the sub-path between these two breakpoints must pass all k triangular regions (w.l.o.g. assume c and d are located on the edges of the triangles). We also assume that the path between s' and t' contains other two breakpoints – we call them a and b – which are on the lower line of the LH (these two breakpoints are also located on the edges of the triangles). For passing these triangles, a path can directly go from a to b. Since the path between s' and t' is optimal in length, the path which contains c and d (π_2) has less than or equal length to the case in which it goes directly from a to b (π_1). If $d_w(\pi_1) = d_w(\pi_2)$, an equivalent path in length which does not contain the horizontal path between c and d exists. If $d_w(\pi_1) < d_w(\pi_2)$, it contradicts our assumption that the path between s and t is optimal in length. For the other case where $d_w(\pi_2) < d_w(\pi_1)$, we consider another path which goes from a to e (a breakpoint on the upper line of the LH and on the edge of the left most triangle) and then from e to f (a breakpoint on the upper line of the LH and on the edge of the right most triangle) and then to b (π_3). According to Lemma 1, since $d_w(\pi_2) < d_w(\pi_1)$, therefore, $d_w(\pi_3) < d_w(\pi_2)$ and this contradicts our assumption that the path between s and t is optimal in length. Thus, the lemma holds. □

Scheduling Three Trains is NP-Complete

Christian Scheffer*

Abstract

We consider the Train Scheduling Problem which can be described as follows: Given m trains via their tracks, i.e., curves in the plane, and the trains' lengths, we want to compute a schedule that moves collision-free and with limited speed the trains along their tracks such that the maximal travel time is minimized. We prove that the Train Scheduling Problem is NP-complete already for three trains.

Furthermore, we extend our NP-completeness construction to the Aircraft Scheduling Problem which means from the case of three trains, i.e., subcurves, to the case of three aircrafts, i.e., disks or squares moving on curves.

1 Introduction

In this paper, we consider a parallel motion planning problem, the Train Scheduling Problem which is naturally motivated from practice and defined as follows: Consider k given *trains* each one defined as a pair which is made up of a curve in the plane, called the *track* of the train and a value, called the *length* of the train. We want to compute a schedule moving collision-free and with bounded velocity all trains along their tracks from their tracks' start points to their tracks' end points such that the maximal travel time called the makespan is minimized.

Furthermore, we consider the Aircraft Scheduling Problem which considers aircrafts, i.e., squares or disks, instead of trains, i.e., subcurves of the tracks.

1.1 Our Results

1. We show that the Train Scheduling Problem is NP-complete already for three trains, see Theorem 1.

2. We establish that the Aircraft Scheduling Problem is NP-complete already for three aircrafts, see Theorem 6.

*Department for Computer Science, Westfälische Wcollision-freeilhelms-Universität Münster
christian.scheffer@uni-muenster.de

1.2 Related Work

Multi-robot coordination is one of the most famous and traditional interfaces between robotics and computational geometry. Due to the amazing large landscape of parallel motion planning topics and corresponding results, we refer to surveys as [7, 8, 9] for detailed overviews.

In their pioneering work, Hopcroft, Schwartz, and Sharir [6] show that even the simple Warehouseman's Problem which requires to coordinate a set of rectangles from a start configuration to a target configuration inside a rectangular box is PSPACE-hard.

In a previous paper [4] accepted to the International Symposium on Computational Geometry 2018, we consider the variant of our Aircraft Scheduling Problem such that the aircrafts' movements are not restricted to curves but to the common Euclidean plane. Amongst others, we showed that this 2D variant is NP-complete for arbitrary many vehicles and gave a constant factor approximation for the case that the aircrafts are sufficiently separated. Furthermore, in [2] we demonstrate a practical realization of our approaches.

In a recent paper [13], we show that there is no FPTAS neither for the Train Scheduling Problem nor for the Aircraft Scheduling Problem but do not answer the question whether there is an efficient algorithm for a constant number of vehicles.

O`Donnell and Lozano-Perez [11] consider the Path Coordination Problem which corresponds to our Aircraft Scheduling Problem and give a $\mathcal{O}(q^2 \log q)$ runtime algorithm for coordinating two robots at which only forward movements are allowed and q is the maximal number of segments on the considered trajectories. Akella and Hutchinson [1] consider Trajectory Coordination Problems in which both the traveling curves and the velocity at which the robots traverse the curves are known. They showed that it is NP-complete to compute departure times for arbitrary many robots such that a minimum-time collision-free robot coordination is achieved.

Reif and Sharir [12] consider Dynamic Movers Problems in which a given polyhedral body B has to be moved collision-free within some 1D, 2D, or 3D space by translations and rotations from a start position to a target position amid a set of obstacles that rotate and move along known trajectories. They provide PSPACE-hardness of the 3D dynamic movement problem if the

body B has to hold a velocity bound and NP-hardness if the body's velocity is unbounded. Furthermore, Reif and Sharir [12] consider Asteroid Avoidance Problems as a special variant of Dynamic Movers Problems in which neither the moving body B nor the obstacles may rotate. In particular, Reif and Sharir provide a near-linear time algorithm for the 1-dimensional Asteroid Avoidance Problem in which each of the obstacles is a polyhedron traveling with fixed (possible distinct) translational velocity along a 1-dimensional line. Reif and Sharir provide an efficient algorithm for the two-dimensional Asteroid Avoidance Problem if the number of the obstacles is a constant and for the three-dimensioal Asteroid Avoidance Problem a single exponential time and a polynomial space algorithm for a convex polyhedron B and arbitrary many obstacles.

2 Preliminaries

A train is a pair (H, L_h) where $L_h \in \mathbb{R}_{>0}$ is the *length* of the train and H is the *track* of the train which is defined as a curve $H : [0,1] \to \mathbb{R}^2$. We simultaneously denote by H, the function $H : [0,1] \to \mathbb{R}^2$ and its image $\{p \in \mathbb{R}^2 \mid \text{ there is a } t \in [0,1] \text{ with } p = H(t)\}$. The length $|T|$ of a track $T : [0,1] \to \mathbb{R}^2$ in the ambient space is defined as its length w.r.t. the Euclidean norm, i.e., $|T| := \int_0^1 ||T'(t)||_2 \, dt$. A *k-fleet* is an k-tuple of trains. Two trains (H, L_h) and (X, L_x) *collide* for the parameters λ_h and λ_x if the subcurves of H and X with midpoints $H(\lambda_h)$ and $X(\lambda_x)$ and lengths L_h and L_x are intersecting each other. A *reparametrization* of a train (H, L_h) is a continuous and piecewise linear function $\alpha : [0, +\infty) \to [0,1]$ such that (1) $\alpha(0) = 0$, (2) there is a minimal value $\lambda \geq 0$ with $\alpha(\mu) = 1$ for all $\mu \geq \lambda$, and (3) the speed of the train is upper-bounded by 1, i.e., for each point in time $t \in [0, +\infty)$, both left and right derivative of $H \circ \alpha$ have Euclidean length at most 1. A schedule for a k-fleet $((T_1, L_1), \ldots, (T_k, L_k))$ is a tuple $(\alpha_1 : [0, M_1] \to [0,1], \ldots, \alpha_k : [0, M_k \to [0,1])$ such that (1) α_i is a reparametrization for the train (T_i, L_i) for all $i \in \{1, \ldots, k\}$ and (2) T_i and T_j do not collide for the parameters $\alpha_i(t)$ and $\alpha_j(t)$ for all $i \neq j \in \{1, \ldots, k\}$ and $t \geq 0$. The *makespan* of the schedule $(\alpha_1 : [0, M_1] \to [0,1], \ldots, \alpha_k : [0, M_k \to [0,1])$ is defined as the maximum $M_{\max}$ of the $M_1, \ldots, M_k$. Wl.o.g., all travel times are equal to $T_{\max}$ by extending α_i with $\alpha_i(t) = \alpha_i(M_i)$ for all $M_i < t < M_{\max}$. Given a k-fleet F, the Train Scheduling Problem asks for a schedule with minimal makespan.

The parameter space P of a k-fleet $((T_1, L_1), \ldots, (T_k, L_k))$ is defined as $P := [0, |T_1|] \times \cdots \times [0, |T_k|]$. The *forbidden* or *black* space $\mathcal{B}$ of P is the union of all parameter points $p = (\lambda_1, \ldots, \lambda_k) \in P$ such that there are two trains T_i and T_j that collide with the parameters λ_i and λ_j. The *allowed* or *white space* $\mathcal{W}$ is defined as $P \setminus \mathcal{B}^\circ$ where $\mathcal{B}^\circ$ denotes the interior of $\mathcal{B}$. Note that the white space $\mathcal{W}$ is closed.

A *path* is a curve $\pi : [0,1] \to P$ and the *length* $\ell(\pi)$ of π is defined as its length w.r.t. the maximum metric, i.e., $\ell(\pi) := \int_0^1 ||\pi'(t)||_\infty \, dt$. An *a-b*-path *in the free space diagram of* $((T_1, L_1), \ldots, (T_k, L_k))$ is a path $\pi \subset \mathcal{W}$ between a and b. If not stated otherwise, a path in the free space diagram is a path $\pi \subset \mathcal{W}$ connecting the points $(0, \ldots, 0)$ and $(|T_1|, \ldots, |T_k|)$.

3 Scheduling Three Vehicles is NP-complete

In this section, we show that surprisingly the Train Scheduling Problem already for three trains, Train (3) for short, and the Aircraft Scheduling Problem already for three aircrafts, Aircraft (3) for short, are NP-complete. We start with the hardness proof for Train (3).

Theorem 1 Train (3) *is NP-complete.*

We show that Train (3) is NP-complete by proving that it is NP-complete to decide whether there is a schedule with a makespan no larger than M where M is an input value. Given an M, w.l.o.g., we set $s := (0,0,0)$ and $t := (M, M, M)$. It is obvious that in an optimal schedule for each point in time there is a train that travels with speed 1. Thus we obtain:

Observation 2 *For a given fleet F, there is a schedule with makespan M if and only if there is an s-t-path of length M w.r.t. the maximum metric in the free space diagram of F.*

In Section 3.1, we construct an instance $\mathcal{I}$ of a 3D-shortest path problem that implies a polynomial time reduction from 3-SAT to a 3D-shortest path problem that is NP-complete. In Section 3.5, we give a reduction of 3-SAT to Train (3) by providing a construction of an instance for Train (3) whose optimal makespan is equal to the shortest path distance of $\mathcal{I}$.

3.1 An NP-Completeness Construction for 3D-Shortest Paths

We consider the three-dimensional Euclidean space $\mathbb{R}^3$ and refer to the three corresponding axes and coordinates as h-, x-, and y-axis and -coordinates. For $a \in \{h, x, y\}$, the *a-length* of a point set $A \subseteq \mathbb{R}^3$ is defined as $\max_{p,q \in A} |p.a - q.a|$ where $p.a$ and $q.a$ denote the a-coordinates of p and q. Furthermore, the *a-distance* between two connected point sets $A, B \subset \mathbb{R}^3$ is defined as $\min_{p \in A, q \in B} |p.a - q.a|$.

Definition 3 *A* plank *is an axis-aligned cuboid $R \subset \mathbb{R}^3$ whose h-, x- or y-length is long enough to be assumed infinity. The* width *and* height *of a plank are the maximum and minimum of the lengths of R in the remaining two axes directions. A plank R is*

- horizontal *if the h- and x-lengths of R are the height of R and infinity*
- vertical *if the h- and y-lengths of R are the height of R and infinity, and*
- perpendicular *if the y- and h-lengths of R are the height of R and infinity.*

The orientation *of R is horizontal, vertical, or perpendicular.*

Next, we define the shortest path problem to which we reduce 3-SAT.

Definition 4 *An instance $\mathcal{I} =: (s, t, L, \xi, \mathcal{R})$ of* 3Dplanks *asks if there is a shortest path of length $L \in \mathbb{R}_{\geq 0}$ w.r.t. the maximum metric between the points $s, t \in \mathbb{R}^3$ and among the set $\mathcal{R}$ of horizontal, vertical, or perpendicular planks that have all a height of ξ.*

For the polynomial-time reduction of 3-SAT to 3Dplanks, we apply the *path encoding technique* as already used for hardness results of other $3D$-shortest path problems [3, 10]. However, in the context of our problem setting we need to ensure important new aspects, see Properties **(P1)**-**(P8)**, because our construction needs to be realisable by the free space diagram of three trains.

In the remainder of this section we show that 3Dplanks is NP-complete. First we prove that 3Dplanks is in NP. After that we give the construction of $\mathcal{I}$ and its analysis.

The piecewise linear environment implies that a shortest path is piecewise linear. Thus, the length of a given path can be calculated within polynomial time w.r.t. the complexity of the environment. Hence, we obtain that 3Dplanks is in NP.

3.2 Outline of the Construction

We consider shortest paths between two points s and t at which st induces a line that has a slope close to 1 w.r.t. x-, y-, and h-coordinate, see Figures 1.

We construct an instance with exponentially many topologically different shortest path classes representing all possible variable assignments for a given 3-SAT formula F. Thus, we first construct a sequence of n *path splitter gadgets*, see the green gadget in Figure 1, at which each path splitter gadget doubles the number of incoming shortest path classes.

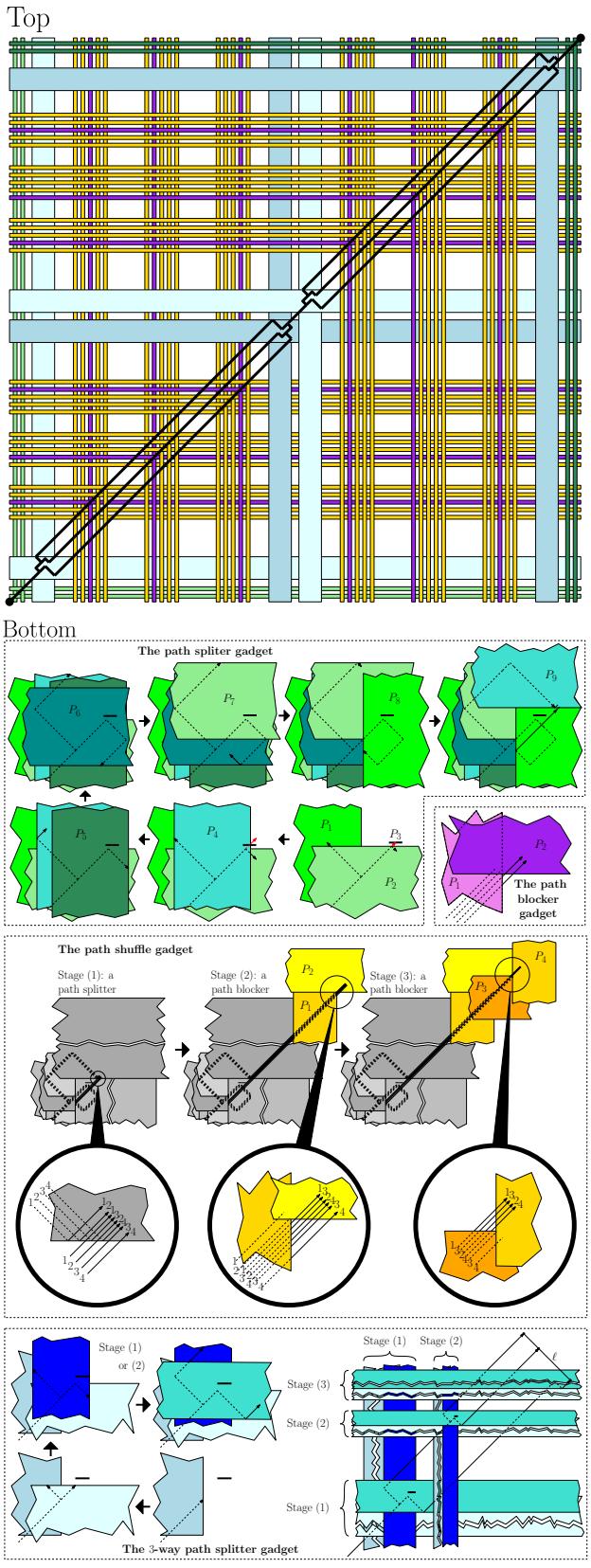


Figure 1: (Top) Construction of $\mathcal{I}$ made up of (3-way) splitter (blue and green), blocker (violet), and shuffle gadgets (orange). (Bottom) Detailed illustration of the used gadgets.

Next it follows a sequence of m clause filters. A clause filter realizes a clause $C_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$ and is made up of three parallel literal filters at which parallel means that each literal filter is passed by an individual tube containing 2^n shortest path classes. In order to produce these tubes, a clause filter starts with a 3-*way path splitter gadget*, see the blue gadget in Figure 1, and ends with an *inverted* 3-way path splitter gadget which merges three input tubes of 2^n shortest path classes into one tube of 2^n shortest path class.

Inside a clause filter C_i, each literal filter represents a literal ℓ_{ij} and is made up of a sequence of n *path shuffle gadgets* (see the orange gadgets in Figure 1) which is interrupted by one *path blocker gadget* (see the violet gadget in Figure 1). The path blocker gadget blocks all shortest path classes whose represented bit assignment for $b_1, \ldots, b_n$ contradicts ℓ_{ij}. In particular, the shortest path classes inside each literal filter lie inside a thin diagonal tube. The prefixed sequence of path shuffle gadgets ensures that all shortest path classes corresponding to bit assignments that contradict ℓ_{ij} lie either on the top left side of the tube or on the bottom right side of the tube. Correspondingly, the path blocker gadgets increases the length of all these shortest path classes to be blocked, i.e., blocks them from being a shortest path of length L between s and t. Finally, the postposed sequence of path shuffle gadget rebuilds the configuration of the shortest path classes inside the tube.

Finally, the bundle of all remaining shortest path classes are merged by a sequence of n inverted path splitting gadgets. By the above discussion it follows that F is satisfiable if and only if there is a shortest path of length L between s and t which we call property **(P1)**.

The first sequence of n path splitter gadgets generates 2^n shortest path classes lying inside a tube of width $\varepsilon \ll 1$ which is maintained for all three copies inside each clause filter.

Each gadget is made up of $\mathcal{O}(1)$ planks, see Figure 1 for an overview and the following section for more details. All in all we have $2n + m(2 + n + 1)$ path gadgets which implies that $\mathcal{I}$ has $\mathcal{O}(mn)$ planks which we call property **(P2)**.

3.3 Detailed Construction of the Path Gadgets

In the following, we discuss the approaches of path splitter gadgets, path blocker gadgets, path shuffle gadgets, and 3-way path splitter gadgets separately, see Figure 1. The inverted versions of the path splitter and the 3-way path splitter gadget are constructed in inverted order.

The input to the path splitter gadget is a thin bundle of shortest path classes, see the green gadget in Figure 1. The produced output is a bundle containing two copies of the input bundle. A perpendicular plank blocks paths from being a shortest path by enforcing the "unwanted" paths to take a detour around the perpendicular plank, see the black bar and the red arrow in the green gadget of Figure 1.

The path blocker gadget blocks either an upper or lower part of the input bundle of shortest paths from being an overall shortest path.

The path shuffle gadget realizes a perfect shuffle to all input shortest path classes and is made up of three stages, see the orange gadget in Figure 1: A path splitter gadget which is colored in gray and two path blocker gadgets colored in yellow, gold, and orange.

The 3-way path splitter gadget produces three instances π_1, π_2, and π_3 of the input shortest path classes for a clause filter representing a clause $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$, see the blue gadget of Figure 1. Each instance π_1, π_2, and π_3 represents one of the three literals $\ell_{i,1}$, $\ell_{i,2}$, and $\ell_{i,3}$ which are logically linked by an "or". Thus, a plank in the clause filter corresponding to C_i is only allowed to have an influence to either the shortest paths in π_1, π_2, or π_3. In order to ensure that, we construct the 3-way path splitter gadget such that the distance between two points from two different bundles of π_1, π_2, and π_3 on a diagonal line ℓ is a constant times larger than the widths of the planks used in the clause filter of C_i, see Figure 1. In particular, the 3-way path splitter gadget is made up of three stages: (1) Four planks, splitting the input shortest path class into two classes, (2) four planks, splitting the upper class of Stage (1) into two shortest path classes π_1 and π_2, and (3) two planks extending the length of the second shortest path class π_3 of Stage (1) about a distance equal to the length extension caused by Stage (2) for π_1 and π_2.

In the following section, we prove that the remaining properties **(P3)**-**(P8)** of Theorem 5 are fulfilled by $\mathcal{I}$.

Theorem 5 3Dplanks *is NP-complete. In particular, for each* 3*-SAT formula* Φ *with* n *variables and* m *clauses, there is an instance* $\mathcal{I} = \mathcal{I}(\Phi) = (s, t, L, \xi, \mathcal{R})$ *of* 3Dplanks *(see Figure 1) such that*

- ***(P1)****: the shortest* s*-*t*-path has a length of* L *if and only if* Φ *is satisfiable,*
- ***(P2)****: there are* $\mathcal{O}(mn)$ *planks,*
- ***(P3)****: all planks have the same height* ξ*,*
- ***(P4)****: the minimal width of a plank is* 1*,*
- ***(P5)****: the minimal* h*-distance of two planks that are not perpendicular is* $\Omega(1)$*,*
- ***(P6)****: all planks have a width of* $\mathcal{O}(mn)$*,*
- ***(P7)****: the maximal* x*-distance between two vertical planks of the same path gadget is* $\mathcal{O}(mn)$*, and*
- ***(P8)****: the maximal* y*-distance between two consecutive horizontal planks of the same path gadget is* $\mathcal{O}(mn)$*.*

3.4 Properties (P3)-(P8)

Property **(P3)** is trivially ensured by explicitly using planks of a common height. Furthermore, w.l.o.g. we assume that the minimal width of planks used in $\mathcal{I}$ is 1 which is property **(P4)**. Otherwise, we scale the whole construction of $\mathcal{I}$ which maintains that all planks have the same height.

For each path gadget, we ensure that the h-distance between two planks that are not perpendicular is $\Omega(1)$. As the input and output path bundles of all path gadgets are diagonal, we can construct $\mathcal{I}$ such that the length of the (shortest) subpath between two consecutive path gadgets is in $\Theta(mn)$. Analogously, we ensure that the shortest path distance between two stages of the same path shuffle or 3-way path splitter gadget is $\Theta(mn)$. Thus we can ensure in our overall construction that the h-distance between any pair of planks that are not perpendicular is at least $\Theta(1)$ which is property **(P5)**.

In our reduction from 3-SAT to 3Dplanks, we apply that some planks are only passed at one side. We guarantee that by choosing the widths of these planks "sufficiently large" (see below for details) such that passing the plank at a forbidden side would cause a detour which prevents the path from being shortest. In the following, we discuss the details of that approach for each type of path gadgets separately.

- The path splitter gadget is constructed such that doubling the input shortest path classes causes a detour of constant length. In order to enforce that a shortest path passes through all $2n$ path splitter gadgets despite a detour of length $\Theta(n)$, we choose the widths of the planks P_1, P_2, P_8, and P_9 of all $2n$ path splitter gadgets as $\Theta(n)$. Furthermore, we choose the widths of the planks P_4, P_5, P_6, and P_7 as $\Theta(1)$ to ensure that a shortest path passes the planks P_4, P_5, P_6, and P_7 on the required sides of the planks, as illustrated in Figure 1.

- A path blocker gadget simply needs to ensure that shortest path classes that represent variable assignments that are forbidden by the represented literal are blocked from being an overall shortest path. Thus, it suffices to choose the width of all planks of all path blocker gadgets as $\Theta(1)$.

- Each path splitter of a path shuffle gadget causes a detour of constant lengths. Inside each clause filter, a shortest path passes through n path shuffle gadgets resulting in summed detour of $\mathcal{O}(n)$. In order to enforce that a shortest path inside each clause filter passes through all n path shuffle gadgets of a literal filter, we choose the widths of the first, the second, and the last two planks of the path splitter part of the path shuffle gadget as $\Theta(n)$. The widths of the remaining planks are chosen equal to the widths of the corresponding planks in the path splitter and path blocker gadgets.

- The 3-way path splitter gadget ensures that the distance between two points from different output bundles is $\Theta(n)$. This results in a detour of length $\Theta(n)$ caused by each 3-way path splitter gadget. In order to ensure that a shortest path passes each plank of a clause filter only at the intended side we choose the width of each one-sided passed plank larger than the entire detour length caused by all m clause filters, i.e., as $\Theta(mn)$.

From the above discussion it follows that the widths of all planks used in our overall construction of $\mathcal{I}$ are upper-bounded by $\mathcal{O}(mn)$ which is property **(P6)**.

Let P_1 and P_2 be two vertical planks of the same path gadget such that there is not another vertical plank lying between P_1 and P_2 w.r.t. the h-axis. We distinguish wether P_1 and P_2 belong to a path splitter gadget or not: If P_1 and P_2 belong to a path splitter gadget, our construction of $\mathcal{I}$ ensures that the x-distance between P_1 and P_2 is 0. As a path splitter gadget is made up of $\mathcal{O}(1)$ planks with widths no larger than $\mathcal{O}(1)$ we obtain that the x-distance between P_1 and P_2 is upper-bounded by $\mathcal{O}(1)$. If P_1 and P_2 belong to path shuffle or a 3-way path splitter gadget, we combine that the path distances between different stages of the path gadget is upper-bounded by $\mathcal{O}(mn)$, that the planks have a width of $\mathcal{O}(mn)$, that each stage is made up of $\mathcal{O}(1)$ planks, and that the path shuffle and the 3-way path splitter gadget are made up of three stages. Thus, we obtain that the x-distance between P_1 and P_2 is upper-bounded by $\mathcal{O}(mn)$. In both cases, we obtain that the x-distance between P_1 and P_2 is upper-bounded by $\mathcal{O}(mn)$ which is property **(P7)**.

A symmetric argument implies that the y-distance between two consecutive horizontal planks belonging to the same path gadget is in $\mathcal{O}(mn)$ which is property **(P8)** concluding the proof of Theorem 5.

3.5 Reduction of 3-SAT to Train (3)

We construct a 3-fleet with optimal makespan equal to the shortest path distance of the instance $\mathcal{I}$ constructed in Section 3.1. A triple $(\lambda_h, \lambda_x, \lambda_y)$ of parameters for the three trains of a 3-fleet F is forbidden if at least two trains collide with their parameters independent from the parameter of the third train. This means the forbidden space $\mathcal{B}$ of F is the union of a set of axis-aligned planks at which each single plank corresponds to an intersection point of two curves, see Figure 2(a)+(b).

The lengths of the planks in the axes directions corresponding to the colliding trains are equal to the lengths of the colliding trains. Furthermore, the plank extends

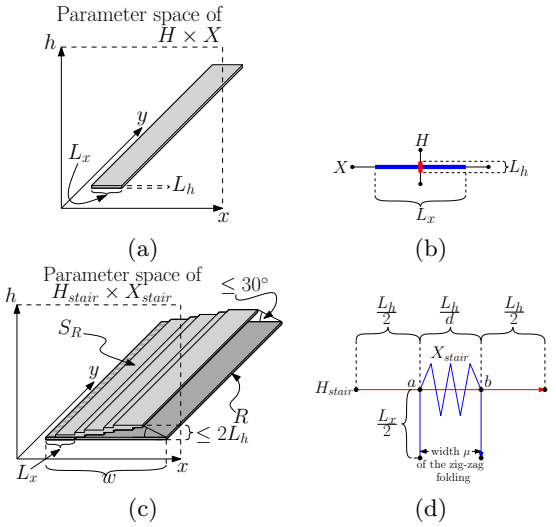


Figure 2: (a) A vertical plank as part caused by an intersection point of the curves H and X that are illustrated in (b). The length of the plank in y-axis direction is infinite because a collision of the trains on H and X is independent from the position of the train on Y. (c) Replacing a vertical plank R by a vertical stairway S_R, and (d) the curves H_{stair} and X_{stair}.

in parallel to the axis corresponding to the third train through the whole parameter space of H, X, and Y. Thus, we occasionally say that a plank has a length of infinity (w.r.t. the axis corresponding to the train which is not necessarily involved in the collision).

The forbidden space of F is piecewise linear implying that a shortest path π' inside the free space diagram is piecewise linear, i.e., π' can be represented by a polynomial sequence of edges it flips over. This implies, that the length of π' can be determined in polynomial time. Thus, Observation 2 implies that Train (3) is in NP.

In order to prove that Train (3) is NP-hard we consider an arbitrary 3-SAT formula Φ and the corresponding instance $\mathcal{I} := \mathcal{I}(\Phi) := (s, t, L, \xi, \mathcal{R})$ of 3Dplanks constructed in Section 3.1. We construct a 3-fleet $F := ((H, L_h), (X, L_x), (Y, L_y))$ and a value L' such that verifying if there is an optimal schedule for F with maximal travel time no larger than L' is equivalent to verifying if there is a shortest path with length no larger than L for $\mathcal{I}$. As the construction of $\mathcal{I}$ induces a polynomial time reduction from 3-SAT to 3Dplanks, it follows that the construction of F induces a polynomial time reduction from 3-SAT to Train (3).

Straightforwardly substituting the planks of $\mathcal{I}$ by planks that are caused by intersection points of the trains (H, L_h), (X, L_x), and (Y, L_y) is not possible because in the construction of $\mathcal{I}$ we use different horizontal planks that have different widths while all horizontal planks in the forbidden space of (H, L_h), (X, L_x), and (Y, L_y) have a width of L_y. Furthermore, there is the same issue with vertical and perpendicular planks. Thus, we replace each single plank R of $\mathcal{I}$ by a so called *stairway* S_R and give an approach how to construct three trains whose forbidden space modulo translations is equal to S_R, see Figures 2(c)+(d) for illustrations.

By assembling all resulting curves corresponding to stairways, we obtain the trains (H, L_H), (X, L_x), and (Y, L_y) concluding the proof of Theorem 1.

3.6 Reduction of 3-SAT to Aircraft (3)

We remark that scheduling three aircrafts, i.e., squares or disks instead of trains, i.e., subcurves is also NP-complete. Generally speaking, we use the 3D-shortest path instance $\mathcal{I}$ and substitute planks of $\mathcal{I}$ by *(curved) wedges*, see Figure 3.

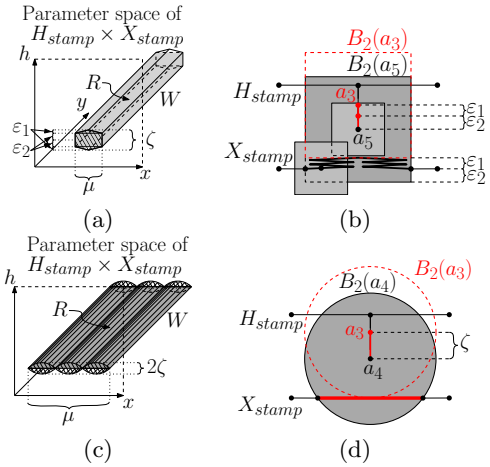


Figure 3: (a)+(c): In the case of square-shaped and disk-shaped aircrafts, we substitute planks by wedges and curved wedges. (b)+(d): In a fixed configuration, two aircrafts collide if and only if the centre of the first aircraft lies inside the square $B_2(c)$ with radius 2 and centre c in the midpoint of the second aircraft.

Theorem 6 Aircraft (3) *is NP-complete for disk-shaped and square-shaped aircrafts.*

4 Conclusion

We presented hardness results for parallel motion planning problems considering objects to moved collision-free along their tracks. Our hardness constructions involve curves that are quite dense in the following manner: Driemel et al. [5] say that a curve is *c-packed* for a $c \geq 0$ if the total intersection of the curve with any ball of radius $r > 0$ is no larger than cr. The curves constructed in our hardness proof are not c-packed for any constant c. Thus, we ask the question whether there is an efficient algorithm for scheduling three trains or aircrafts along c-packed curves.

References

[1] S. Akella and S. Hutchinson. Coordinating the motions of multiple robots with specified trajectories. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, May 11-15, 2002, Washington, DC, USA*, pages 624–631, 2002.

[2] A. T. Becker, S. P. Fekete, P. Keldenich, L. Lin, and C. Scheffer. Coordinated motion planning: The video. In C. Tóth and B. Speckmann, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 74:1–74:6, 2018. Video available at https://youtu.be/0OrG72sX5gk.

[3] J. F. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 49–60, 1987.

[4] E. D. Demaine, S. P. Fekete, P. Keldenich, H. Meijer, and C. Scheffer. Coordinated Motion Planning: Coordinating a Swarm of Labeled Robots with Bounded Stretch. In C. Tóth and B. Speckmann, editors, *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:15, 2018.

[5] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the fréchet distance for realistic curves in near linear time. *Discret. Comput. Geom.*, 48(1):94–127, 2012.

[6] J. E. Hopcroft, J. T. Schwartz, and M. 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.

[7] L. E. Kavraki and S. M. LaValle. Motion planning. In *Springer Handbook of Robotics*, pages 139–162. 2016.

[8] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[9] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[10] J. S. B. Mitchell and M. Sharir. New results on shortest paths in three dimensions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 124–133, 2004.

[11] P. A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation, Scottsdale, Arizona, USA, May 14-19, 1989*, pages 484–489, 1989.

[12] J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. *J. ACM*, 41(4):764–790, 1994.

[13] C. Scheffer. Train scheduling: Hardness and algorithms. In *Proceedings of the 14th International Conference on Algorithms and Computation (WALCOM)*, pages 342–347, 2020.

Chasing Puppies

Jeff Erickson*

I will describe a topological solution to the following puzzle, which Michael Biro posed as an open problem at CCCG 2013. A human and a puppy find themselves at different points on a walking trail, which is a simple closed curve in the plane. The human and puppy can see each other from anywhere on the trail, but they cannot leave the trial. The puppy always moves as quickly as possible to decrease its distance to the human. Can the human catch the puppy? (Yes!)

This is joint work with Irina Kostitsyna, Maarten Löffler, Tillman Miltzow, Jérôme Urhausen, and Jordi Vermeulen.

*University of Illinois at Urbana-Champaign, USA, jeffe@illinois.edu

Folding Small Polyominoes into a Unit Cube

Kingston Yao Czajkowski* Erik D. Demaine† Martin L. Demaine† Kim Eppling‡ Robby Kraft§
Klara Mundilova† Levi Smith¶

Abstract

We demonstrate that a 3×3 square can fold into a unit cube using horizontal, vertical, and diagonal creases on the 6×6 half-grid. Together with previous results, this result implies that all tree-shaped polyominoes with at least nine squares fold into a unit cube. We also make partial progress on the analogous problem for septominoes and octominoes by showing a half-grid folding of the U septomino and 2×4 rectangle into a unit cube.

1 Introduction

Which polyominoes fold into a unit cube? Aichholzer et al. [ABD+18] introduced this problem at CCCG 2015, along with a variety of different models for folding. Table 1 summarizes the main models and known results. We focus here on the powerful ***half-grid model*** (the bottom two rows of Table 1) where

1. the polyomino can be folded along horizontal, vertical, and $\pm 45°$ diagonal creases;
2. every crease has endpoints whose coordinates are integer multiples of $\frac{1}{2}$; and
3. each crease can be folded by $\pm 90°$ or $\pm 180°$.

In particular, the paper can overlap itself, using multiple layers to cover the cube (as in origami, but unlike polyhedron unfolding), so long as the paper covers every point of the cube. Look ahead to Figures 4, 5, and 6 for examples of foldings in the half-grid model.

A strong positive result [ABD+18, Theorem 3] is that every polyomino of at least ten squares can fold into a unit cube in the half-grid model.[1] In this paper, we tackle the analogous problem for smaller polyominoes, with at most nine squares.[2]

Any polyomino folding into a unit cube has at least six squares (because the cube has surface area 6). Indeed, for hexominoes, the half-grid and diagonal features of the model are not useful, because the folding cannot have any overlap (again by an area argument). Therefore, the hexominoes that fold into a unit cube are exactly the eleven hexomino nets of the cube; see e.g. Gardner [Gar89] for the list. Aichholzer et al. [ABD+18, Fig. 16] verified by exhaustive search that this claim remains true even if we allow cutting the polyomino with slits until the dual graph (with a vertex for each square and edges for uncut edge adjacency) is a tree; we call these ***tree-shaped polyominoes***.

In between this solved hexomino case and the universally foldable $\geq$ 10-ominoes are polyominoes with between seven and nine squares: septominoes, octominoes, and nonominoes. For these cases, Aichholzer et al. [ABD+18, Fig. 17] did an exhaustive enumeration of which *tree-shaped* polyominoes cannot fold into a unit cube in a more restrictive ***grid + diagonals model*** (the two middle rows of Table 1), which is identical to the half-grid model above except that every crease has endpoints whose coordinates are integers.

Therefore the only remaining unsolved tree-shaped cases for the half-grid model are exactly these examples not foldable in the grid + diagonals model. Aichholzer et al. [ABD+18, Fig. 17] lists twelve septominoes, three octominoes, and just one nonomino with the property that some cutting into a tree-shaped polyomino has no grid folding. Figures 1, 2, and 3 list all tree-shaped cuttings of these polyominoes that lack a grid folding, as computed by Aichholzer for [ABD+18] (but which have not previously appeared).

2 Results

In this paper, we show how to fold the one nonomino case (the 3×3 square), one of the octomino cases (the 2×4 square), and one of the septomino cases (the U)

*Cairo-Durham Middle School, Cairo, NY, USA. stonkinge41@gmail.com

†MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA. {edemaine,mdemaine,kmundil}@mit.edu

‡Massachusetts Institute of Technology, Cambridge, MA, USA. kimeppling@gmail.com

§The New School, New York, NY, USA. robbykraft@gmail.com

¶The Newton School, Strafford, VT, USA. levipaulsmith08@gmail.com

[1] A small typo in [ABD+18] is that the Introduction fails to mention the half-grid nature of this result, though their Theorem 3 correctly states the result. Their result also guarantees that every face of the cube is covered by a seamless square in the folding, a property we ignore here.

[2] The Introduction of [ABD+18] claims to "characterize all the polyominoes that can be folded into a unit cube, in grid-based models", but in fact the characterizations for < 10 and ≥ 10 squares are in two different models, as we now detail, so neither is a complete characterization.

Coordinates	Creases	Polyominoes	Polyomino sizes and results
Grid	Orthogonal	Tree-shaped	Characterized ≤ 14 [ABD$^+$18] Characterized height-2 and height-3 [ABD$^+$18] OPEN: ≥ 15 of height ≥ 3
Grid	Orthogonal	Arbitrary	Partially characterized [AAC$^+$19] OPEN: ≥ 7
Grid	Diagonal	Tree-shaped	Characterized ≤ 14; all $10, 11, 12, 13, 14$ [ABD$^+$18] OPEN: ≥ 15
Grid	Diagonal	Arbitrary	OPEN: ≥ 7
Half-grid	Diagonal	Tree-shaped	**All ≥ 9 [this paper]** OPEN: $7, 8$
Half-grid	Diagonal	Arbitrary	All ≥ 10 [ABD$^+$18] OPEN: $7, 8, 9$

Table 1: Summary of known/open characterizations of which polyominoes fold into a unit cube in six different models, according to whether crease endpoints must be integers ("grid") or can be half-integers ("half-grid"); whether creases must be horizontal and vertical ("orthogonal") or they can also be at $\pm 45°$ ("diagonal"); and whether the polyominos' duals must be trees ("tree-shaped") or can have cycles ("arbitrary"). Numbers (between 7 and 15) refer to the number of squares in the polyomino, except that "height" refers to the smaller dimension of the polyomino's bounding box. "All" means that all polyominoes of a given size fold into a unit cube; "Characterized" means that there is a list of which do and which do not; "Partially characterized" means that there are necessary conditions and sufficient conditions.

into a cube in the half-grid model. Because our foldings do not require any particular cuts, they also work for any tree-shaped polyomino resulting from cutting these polyominoes (the shaded cases in Figures 1, 2, and 3). In particular, our solution to the sole nonomino case implies (together with [ABD$^+$18, Theorem 3]) that all tree-shaped polyominoes with at least nine squares fold into a cube in the half-grid model.

Figures 4, 5, and 6 show how to fold a 3×3 square, 2×4 rectangle, and U, respectively, into a unit cube. In the crease patterns of subfigures (a), dotted lines indicate the integer grid, while solid lines indicate creases and paper boundary. Mountain creases are drawn in red, valley creases are drawn in blue, and crease lines are partially transparent if they fold by $\pm 90°$ and fully opaque if they fold by $\pm 180°$. This notation enables verification of the folding via Origami Simulator [GDG18], which generated the intermediate 3D foldings in subfigures (c). (As Figure 6(c) makes clear, the simulation allows collisions and material stretch during the motion, but it still verifies the final folding.) Subfigures (b) present human-drawn views of the folded states with the faces spread out slightly to make clear how the faces can be stacked while avoiding collision. In addition to the full folded state with translucent faces (right), which reveals mainly the front three faces of the cube, we show the subfolding of just the back three faces of the cube (left). To show the correspondence between the crease pattern (a) and folded state (b), we also label the faces that make up the outer cube surface.

The foldings in Figures 4 and 5 shift the grid of the polyomino by $\frac{1}{2}$ to make the grid of the unit cube. Curiously, the folding of the U septomino in Figure 6 does not, and barely uses the half-grid model by having two crossing diagonals which meet at a half-integer point.

3 Other Related Work

Beyond the problem studied in this paper, several other variations have been considered.

In addition to the results mentioned above, Aichholzer et al. [ABD$^+$18] studied the ***grid model*** where creases must be horizontal or vertical (no diagonals) and have endpoints at integer coordinates (the top two rows of Table 1). Specifically, they characterized exactly which tree-shaped polyominoes of height 2 or 3 (i.e., fitting in a $2 \times \infty$ or $3 \times \infty$ strip) fold into a unit cube; their condition can be checked in linear time. A general characterization remains open. They also proved separations between the models in Table 1, along with a few other models, and characterized which polyiamonds (edge-to-edge joinings of equilateral triangles) fold into a unit tetrahedron in an analog to the grid model.

At CCCG 2019, Aichholzer et al. [AAC$^+$19] considered the grid model when the polyomino has holes (and is thus not tree-shaped). This case corresponds to some puzzles invented by Nikolai Beluhov [Bel14] which originally motivated [ABD$^+$18] as well. Aichholzer et al. [AAC$^+$19] gave sufficient conditions when a polyomino containing certain hole shapes can fold into a cube, as well as some necessary conditions, but a general characterization remains open.

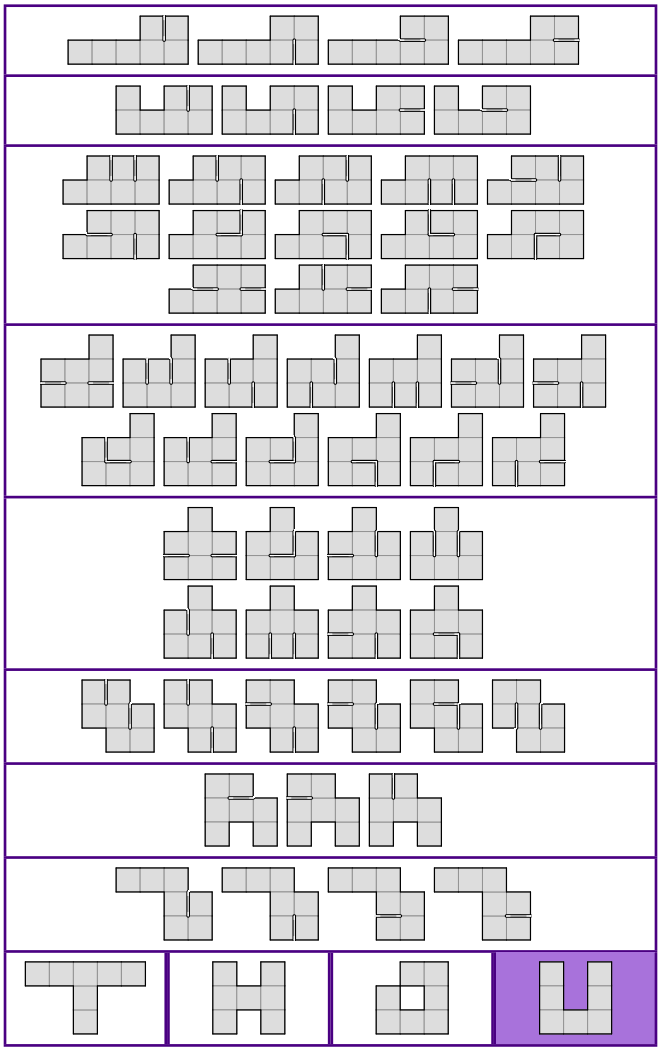


Figure 1: Tree-shaped septominoes that cannot fold into a unit cube in the grid + diagonals model, as computed by Aichholzer [ABD+18]. Figure 6 shows how to fold the shaded case in the half-grid model.

Gardner [Gar95] posed a puzzle about cutting the 3×3 square with slits and then folding along orthogonal grid lines into a unit cube with the additional property of just one side of the paper showing on the outside. Gardner gave one solution, and stated that it can be done "in many different ways". Dunham and Whieldon [DW17] subsequently found all solutions (with and without the additional property) by exhaustive search.

Off the polyomino grid, Catalano-Johnson, Loeb, and Beebee [CLB01] (see also [DO07, Section 15.4.1]) proved that the smallest square that folds into a unit cube has dimensions $(2\sqrt{2}) \times (2\sqrt{2}) \approx 2.8284 \times 2.8284$. This folding implies a folding of a 3×3 square into a unit cube: just fold away the extra material first. Our innovation is to show that there is a folding in the half-grid model. By contrast, the solution in [CLB01] rotates the square 45° to make the grid of the unit cube.

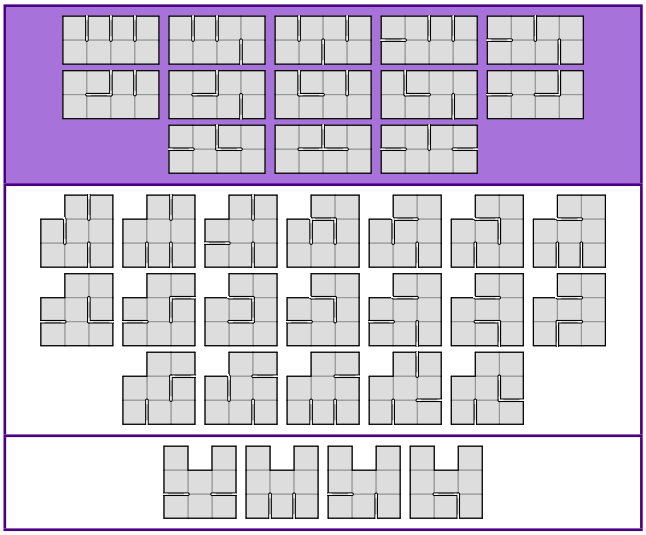


Figure 2: Tree-shaped octominoes that cannot fold into a unit cube in the grid + diagonals model, as computed by Aichholzer [ABD+18]. Figure 5 shows how to fold the shaded cases in the half-grid model.

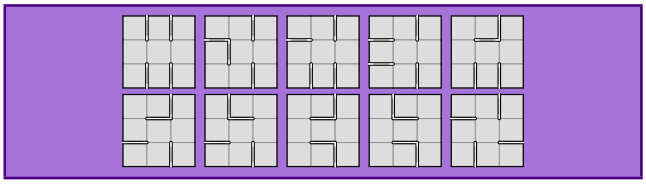


Figure 3: Tree-shaped nonominoes that cannot fold into a unit cube in the grid + diagonals model, as computed by Aichholzer [ABD+18]. Figure 4 shows how to fold all of them in the half-grid model.

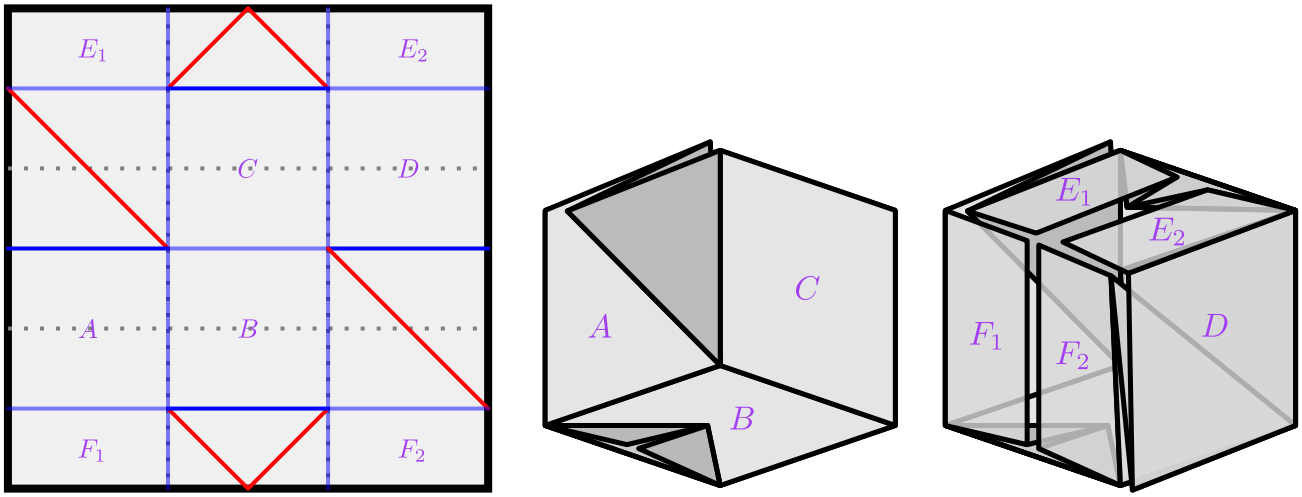
4 Open Problems

We conjecture that the remaining septomino and octomino cases cannot be folded into a cube, but could not find an easy argument for impossibility. The best approach may be an exhaustive search for foldings in the half-grid model.

Beyond just tree-shaped polyominoes, we conjecture that all polyominoes with at least nine squares fold into a cube in the half-grid model. The result for at least ten squares [ABD+18, Theorem 3] does not rely on the tree-shaped property, but the existence of grid foldings for all nonominoes beyond the 3×3 square does. Thus we would need to verify that all 438 non-tree-shaped nonominoes fold into a cube, which we have started to do, but may be easiest to complete via exhaustive search.

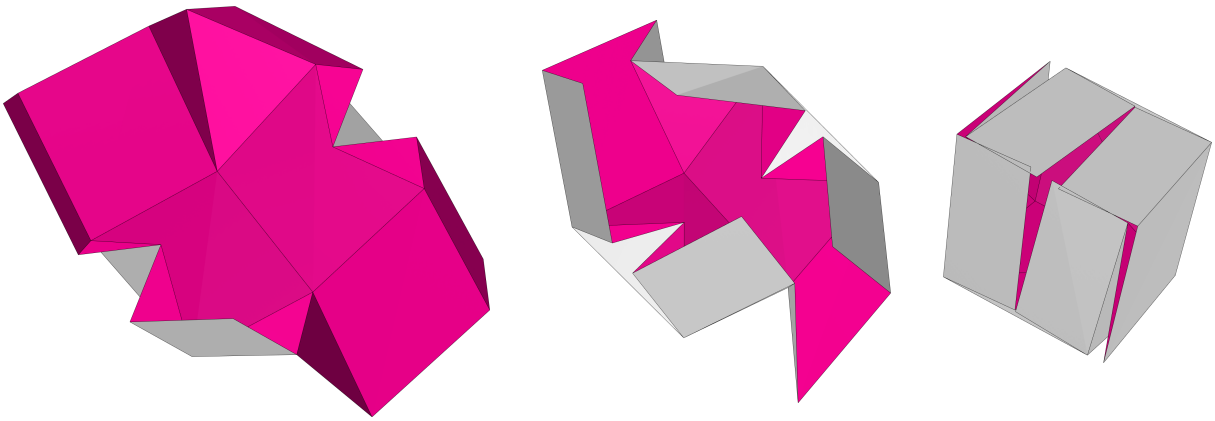
There are also countless other models and additional conditions to consider. We mention a few now.

As mentioned in Footnote 1, the universal folding for at least ten squares [ABD+18, Theorem 3] guarantees that every face of the cube is covered by a seamless unit



(a) Crease pattern

(b) Folded state (right) and back faces (left)



(c) Folding animation from Origami Simulator [GDG18]

Figure 4: Folding the 3×3 square into a unit cube.

square. Our folding of the U septomino in Figure 6 shares this property, but we conjecture that this property is unattainable for the 2×4 rectangle or 3×3 square because (unlike the U) they seem to need to misalign the cube's grid with the polyomino's grid.

Gardner's 3×3 puzzle [Gar95] mentioned in Section 3 required that the surface of the cube be made entirely from the same side of the piece of paper. Our foldings of the 3×3 square (Figure 4) and 2×4 rectangle (Figure 5), while our folding of the U (Figure 6) does not, and we conjecture that it cannot. With this restriction, many other problems become open again. For example, can all polyominoes of at least ten squares still fold into a unit cube?

Finally, the animations we draw in Figures 4(c), 5(c), and 6(c) raise the question of which cube foldings are achievable as ***rigid origami*** (avoiding collisions while folding only at creases). This direction has yet to be explored.

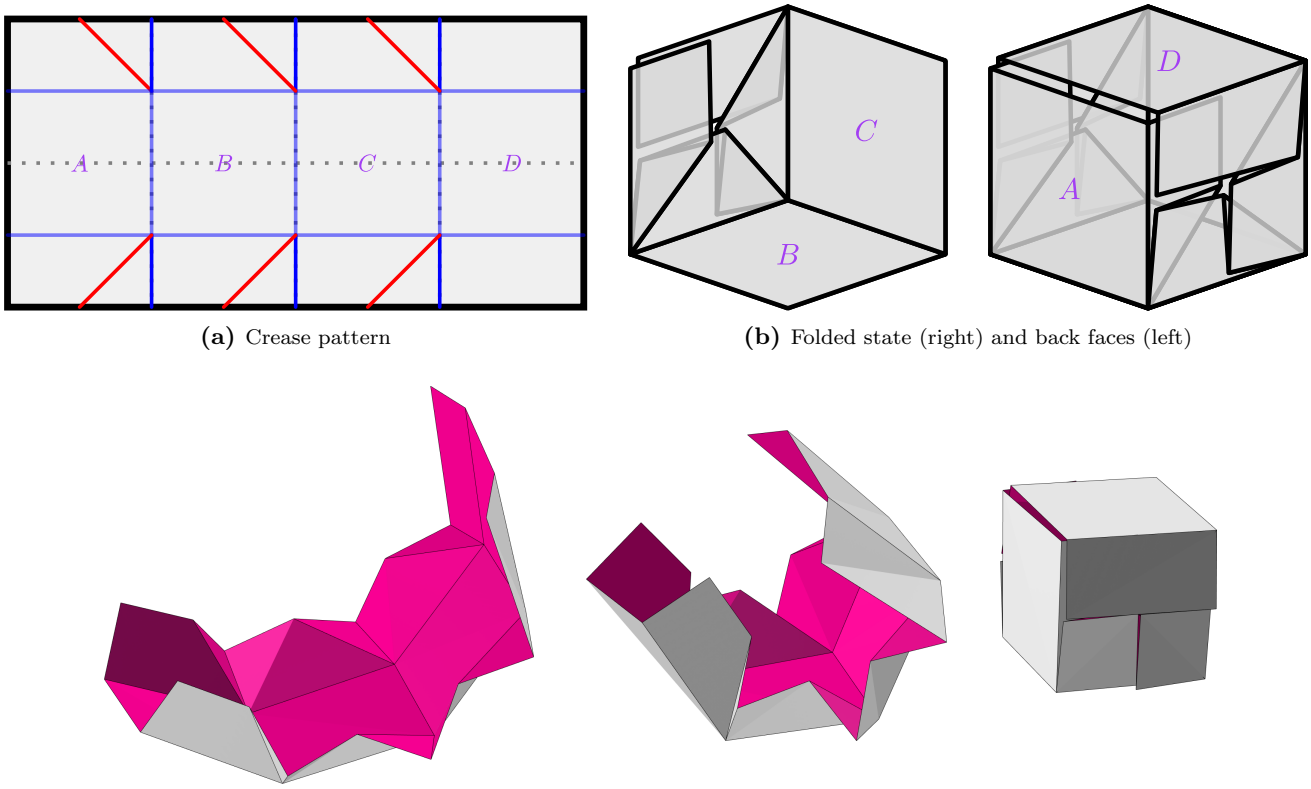
Acknowledgments

This work was initiated at the 9th Annual OrigaMIT Convention held at MIT on November 9, 2019, where multiple groups tackled the then-unsolved puzzles posed in [DD19], which are now solved by Figure 4.

We thank Oswin Aichholzer for providing the data from [ABD+18] that enabled us to draw Figures 1, 2, and 3.

References

[AAC+19] Oswin Aichholzer, Hugo A. Akitaya, Kenneth C. Cheung, Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Linda Kleist, Irina Kostitsyna, Maarten Löffler, Zuzana Masárová, Klara Mundilova, and Christiane Schmidt. Folding polyominoes with holes into a cube. In *Proceedings of the 31st Canadian Conference in Computational Geome-*



(a) Crease pattern

(b) Folded state (right) and back faces (left)

(c) Folding animation from Origami Simulator [GDG18]

Figure 5: Folding the 2×4 rectangle into a unit cube.

try (CCCG 2019), pages 164–170, Edmonton, Alberta, Canada, August 2019.

[ABD+18] Oswin Aichholzer, Michael Biro, Erik D. Demaine, Martin L. Demaine, David Eppstein, Sándor P. Fekete, Adam Hesterberg, Irina Kostitsyna, and Christiane Schmidt. Folding polyominoes into (poly)cubes. *International Journal of Computational Geometry and Applications*, 28(3):197–226, 2018. Originally at CCCG 2015.

[Bel14] Nikolai Beluhov. Cube folding. https://nbpuzzles.wordpress.com/2014/06/08/cube-folding/, 2014.

[CLB01] Michael L. Catalano-Johnson, Daniel E. Loeb, and John Beebee. A cubical gift: 10716. *The American Mathematical Monthly*, 108(1):81–82, 2001.

[DD19] Erik D. Demaine and Martin L. Demaine. Cube folding puzzles: Origamit 2019 edition. http://erikdemaine.org/puzzles/CubeFolding/OrigaMIT2019/, 2019.

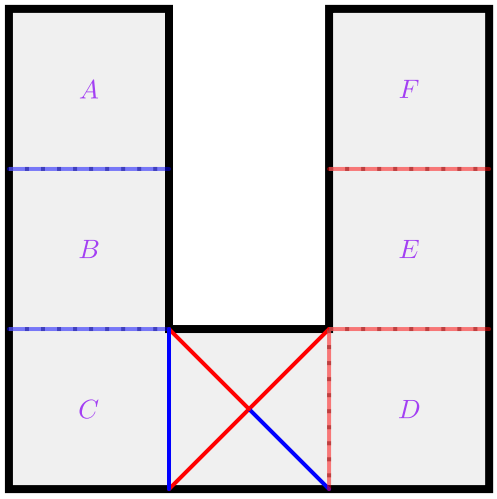
[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007.

[DW17] Jill Bigley Dunham and Gwyneth R. Whieldon. Enumeration of solutions to Gardner's paper cutting and folding problem. In *The Mathematics of Various Entertaining Subjects*, volume 2, pages 108–124. Princeton University Press, 2017.

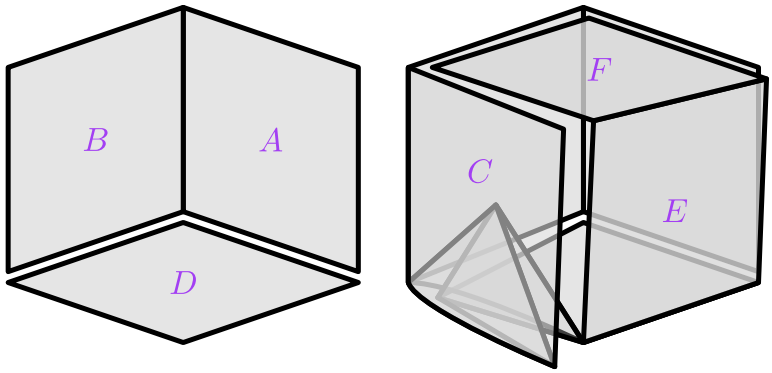
[Gar89] Martin Gardner. Hypercubes. In *Mathematical Carnival*, chapter 4, pages 41–54. The Mathematical Association of America, Washington, D.C., 1989.

[Gar95] Martin Gardner. Paper cutting. In *New Mathematical Diversions*, chapter 5, pages 58–69. The Mathematical Association of America, Washington, D.C., 1995.

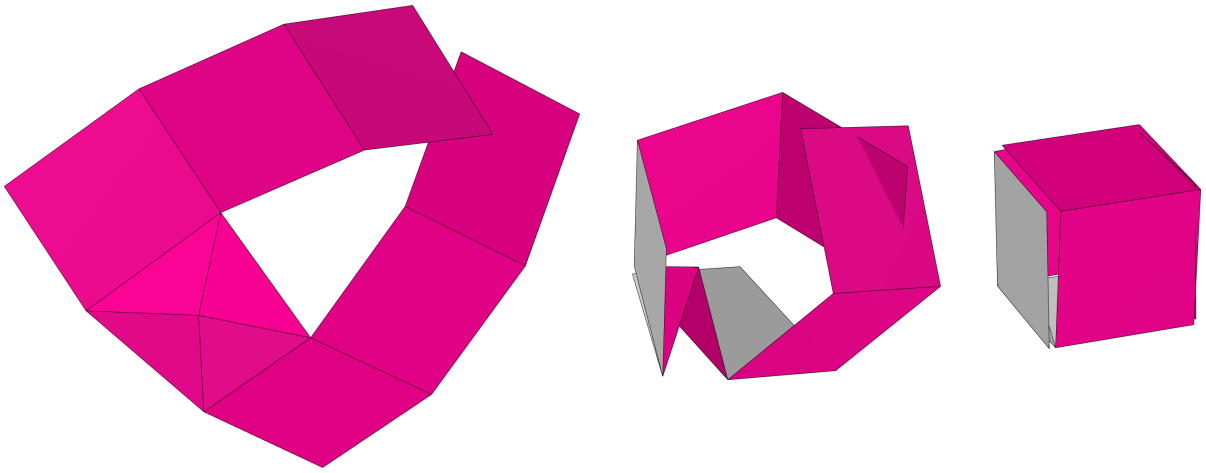
[GDG18] Amanda Ghassaei, Erik D. Demaine, and Neil Gershenfeld. Fast, interactive origami simulation using GPU computation. In *Origami^7: Proceedings of the 7th International Meeting on Origami in Science, Mathematics and Education (OSME 2018)*, volume 4, pages 1151–1166. Oxford, England, September 2018. https://origamisimulator.org.



(a) Crease pattern



(b) Folded state (right) and back faces (left)



(c) Folding animation from Origami Simulator [GDG18]

Figure 6: Folding the U septomino into a unit cube.

Some Polycubes Have No Edge Zipper Unfolding

Erik D. Demaine* Martin L. Demaine* David Eppstein† Joseph O'Rourke‡

Abstract

It is unknown whether every polycube (polyhedron constructed by gluing cubes face-to-face) has an edge unfolding, that is, cuts along edges of the cubes that unfolds the polycube to a single nonoverlapping polygon in the plane. Here we construct polycubes that have no ***edge zipper unfolding*** where the cut edges are further restricted to form a path.

1 Introduction

A ***polycube*** P is an object constructed by gluing cubes whole-face to whole-face, such that its surface is a manifold. Thus the neighborhood of every surface point is a disk; so there are no edge-edge nor vertex-vertex nonmanifold surface touchings. Here we only consider polycubes of genus zero. The ***edges*** of a polycube are all the cube edges on the surface, even when those edges are shared between two coplanar faces. Similarly, the ***vertices*** of a polycube are all the cube vertices on the surface, even when those vertices are ***flat***, incident to 360° total face angle. Such polycube flat vertices have degree 4. It will be useful to distinguish these flat vertices from ***corner vertices***, nonflat vertices with total incident angle $\neq 360°$ (degree 3, 5, or 6). For a polycube P, let its **1-*skeleton graph*** G_P include every vertex and edge of P, with vertices marked as either corner or flat.

It is an open problem to determine whether every polycube has an ***edge unfolding*** (also called a ***grid unfolding***) — a tree in the 1-skeleton that spans all corner vertices (but need not include flat vertices) which, when cut, unfolds the surface to a ***net***, a planar nonoverlapping polygon [O'R19]. By ***nonoverlapping*** we mean that no two points, each interior to a face, are mapped to the same point in the plane. This definition allows two boundary edges to coincide in the net, so the polygon may be "weakly simple." The intent is that we want to be able to cut out the net and refold to P.

It would be remarkable if every polycube could be edge unfolded, but no counterexample is known. There has been considerable exploration of orthogonal polyhedra, a more general type of object, for which there are examples that cannot be edge-unfolded [BDD+98]. (See [DF18] for citations to earlier work.) But polycubes have more edges in their 1-skeleton graphs for the cut tree to follow than do orthogonal polyhedra, so it is conceivably easier to edge-unfold polycubes.

A restriction of edge unfolding studied in [She75, DDL+10, O'R10, DDU13] is ***edge zipper unfolding*** (also called ***Hamiltonian unfolding***). A ***zipper*** unfolding has a cut tree that is a path (so that the surface could be "unzipped" by a single zipper). It is apparently unknown whether even the highly restricted edge zipper unfolding could unfold every polycube to a net. The result of this note is to settle this question in the negative: polycubes are constructed none of which have an edge zipper unfolding. Two polycubes in particular, shown in Fig. 1, have no such unfolding. Other polycubes with the same property are built upon these two.

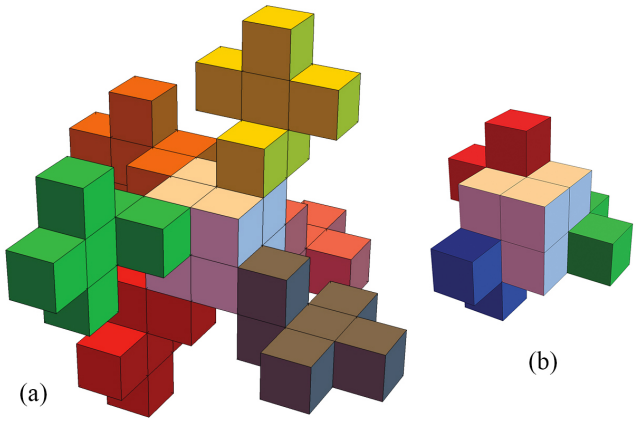


Figure 1: Two polycubes that have no edge zipper unfolding.

2 Hamiltonian Paths

Shephard [She75] introduced Hamiltonian unfoldings of convex polyhedra, what we refer to here as edge zipper unfolding, following the terminology of [DDL+10]. Any edge zipper unfolding must cut along a Hamiltonian path of the vertices. It is easy to see that not every convex polyhedron has an edge zipper unfolding, simply because the rhombic dodecahedron has no Hamiltonian

*MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139, USA, {edemaine,mdemaine}@mit.edu

†Computer Science Department University of California, Irvine, CA 92679, USA eppstein@uci.edu Supported in part by NSF grants CCF-1618301 and CCF-1616248

‡Department of Computer Science, Smith College, Northampton, MA 01063, USA. jorourke@smith.edu.

path. This counterexample avoids confronting the difficult nonoverlapping condition.

We follow a similar strategy here, constructing a polycube with no Hamiltonian path. But there is a difference in that a polycube edge zipper unfolding need not include flat vertices, and so need not be a Hamiltonian path in G_P. Thus identifying a polycube P that has no Hamiltonian path does not immediately establish that P has no edge zipper unfolding, if P has flat vertices.

So one approach is to construct a polycube P that has no flat vertices—every vertex is a corner vertex. Then, if P has no Hamiltonian path, then it has no edge zipper unfolding. A natural candidate is the polycube object P_6 shown in Fig. 2. However, the 1-skeleton of P_6 does

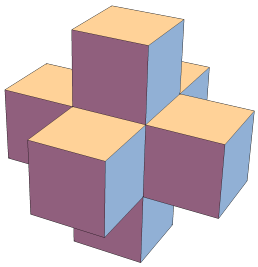


Figure 2: All of P_6's vertices are corner vertices.

admit Hamiltonian paths, and indeed we found a path that unfolds P_6 to a net.

Let $\overline{G}_P$ be the dual graph of P: each cube is a node, and two nodes are connected if they are glued face-to-face. A ***polycube tree*** is a polycube whose dual graph is a tree. P_6 is a polycube tree. That it has a Hamiltonian path is an instance of a more general claim:

Lemma 1 *The graph G_P for any polycube tree P has a Hamiltonian cycle.*

Proof. It is easy to see by induction that every polycube tree can be built by gluing cubes each of which touches just one face at the time of gluing: never is there a need to glue a cube to more than one face of the previously built object.

A single cube has a Hamiltonian cycle. Now assume that every polycube tree of $\leq n$ cubes has a Hamiltonian cycle. For a tree P of $n+1$ cubes, remove a $\overline{G}_P$ leaf-node cube C, and apply the induction hypothesis. The exposed square face f to which C glues to make P includes either 2 or 3 edges of the Hamiltonian cycle (4 would close the cycle; 1 or 0 would imply the cycle misses some vertices of f). It is then easy to extend the Hamiltonian cycle to include C, as shown in Fig. 3. □

So to prove that a polycube tree has no edge zipper unfolding would require an argument that confronted nonoverlap. This leads to an open question:

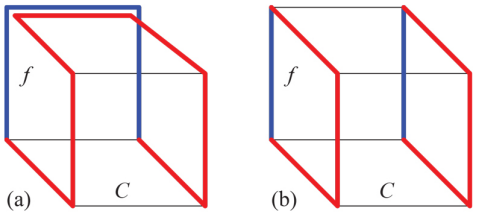


Figure 3: (a) f contains 3 edges of the cycle (blue); (b) f contains 2 edges of the cycle. The cycles are extended to C by replacing the blue with the the red paths.

Question 1 *Does every polycube tree have an edge zipper unfolding?*

3 Bipartite G_P

To guarantee the nonexistence of Hamiltonian paths, we can exploit the bipartiteness of G_P, using Lemma 3 below.

Lemma 2 *A polycube graph G_P is 2-colorable, and therefore bipartite.*

Proof. Label each lattice point p of $\mathbb{Z}^3$ with the $\{0,1\}$-parity of the sum of the Cartesian coordinates of p. A polycube P's vertices are all lattice points of $\mathbb{Z}^3$. This provides a 2-coloring of G_P; 2-colorable graphs are bipartite. □

The ***parity imbalance*** in a 2-colored (bipartite) graph is the absolute value of the difference in the number of nodes of each color.

Lemma 3 *A bipartite graph G with a parity imbalance > 1 has no Hamiltonian path.*[1]

Proof. The nodes in a Hamiltonian path alternate colors 010101.... Because by definition a Hamiltonian path includes every node, the parity imbalance in a bipartite graph with a Hamiltonian path is either 0 (if of even length) or 1 (if of odd length). □

So if we can construct a polycube P that (a) has no flat vertices, and (b) has parity imbalance > 1, then we will have established that P has no Hamiltonian path, and therefore no edge zipper unfolding. We now show that the polycube P_{44}, illustrated in Fig. 4, meets these conditions.

Lemma 4 *The polycube P_{44}'s graph $G_{P_{44}}$ has parity imbalance of 2.*

Proof. Consider first the $2\times2\times2$ cube that is the core of P_{44}; call it P_{222}. The front face F has an extra 0; see Fig. 5. It is clear that the 8 corners of P_{222} are all

[1] Stated at http://mathworld.wolfram.com/HamiltonianPath.html.

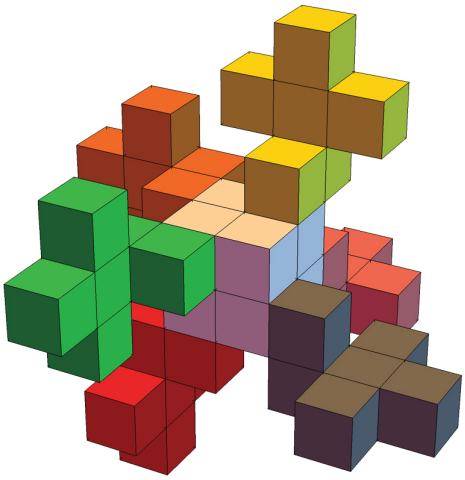


Figure 4: The polycube P_{44}, consisting of 44 cubes, has no Hamiltonian path.

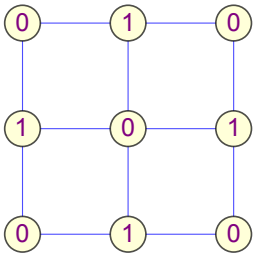


Figure 5: 2-coloring of one face of P_{222}.

colored 0. The midpoint vertices of the 12 edges of P_{222} are colored 1. Finally the 6 face midpoints are colored 0. So 14 vertices are colored 0 and 12 colored 1.

Next observe that attaching a cube C to exactly one face of any polycube does not change the parity: the receiving face f has colors 0101, and the opposite face of C has colors 1010.

Now, P_{44} can be constructed by attaching six copies of a 6-cube "cross," call it P_+, which in isolation is a polycube tree and so can be built by attaching cubes each to exactly one face. And each P_+ attaches to one corner cube of P_{222}. Therefore P_{44} retains P_{222}'s imbalance of 2. □

The point of the P_+ attachments is to remove the flat vertices of P_{222}. Note that when attached to P_{222}, each P_+ has only corner vertices.

Theorem 5 *Polycube P_{44} has no edge zipper unfolding.*

Proof. Although it takes some scrutiny of Fig. 4 to verify, P_{44} has no (degree-4) flat vertices. Thus an edge zipper unfolding must pass through every vertex, and so be a Hamiltonian path. Lemma 4 says that $G_{P_{44}}$ has imbalance 2, and Lemma 3 says it therefore cannot have a Hamiltonian path. □

4 Construction of P_{14}

It turns out that the smaller polycube P_{14} shown in Fig. 6 also has no edge zipper unfolding, even though it has flat vertices. To establish this, we still need an

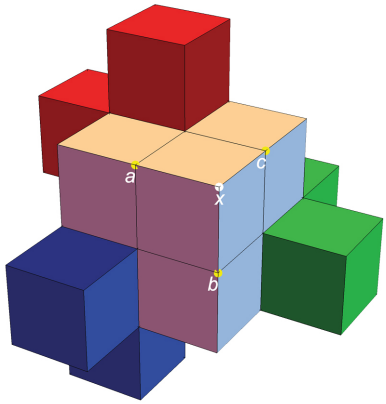


Figure 6: P_{14}: P_{222} with six 1-cube attachments.

imbalance > 1, which easily follows just as in Lemma 4:

Lemma 6 *The polycube P_{14}'s graph $G_{P_{14}}$ has parity imbalance of 2.*

But notice that P_{14} has three flat vertices: a, b, and c.

Theorem 7 *Polycube P_{14} has no edge zipper unfolding.*

Proof. An edge zipper unfolding need not pass through the three flat vertices, a, b, and c, but it could pass through one, two, or all three. We show that in all cases, an appropriately modified subgraph of $G_{P_{14}}$ has no Hamiltonian path. Let ρ be a hypothetical edge zipper unfolding cut path. We consider four exhaustive possibilities, and show that each leads to a contradiction.

(0) **ρ includes a, b, c.** So ρ is a Hamiltonian path in $G_{P_{14}}$. But Lemma 6 says that $G_{P_{14}}$ has imbalance 2, and Lemma 3 says that no such graph has a Hamiltonian path.

(1) **ρ excludes one flat vertex a and includes b, c.** (Because of the symmetry of P_{14}, it is no loss of generality to assume that it is a that is excluded.) If ρ excludes a, then it does not travel over any of the four edges incident to a. Thus we can delete a from $G_{P_{14}}$; say that $G_{-a} = G_{P_{14}} \setminus a$. This graph is shown in Fig. 7. Following the coloring in Fig. 5, all corners of P_{222} are colored 0, so each of the edge midpoints a, b, c is colored 1. The parity imbalance

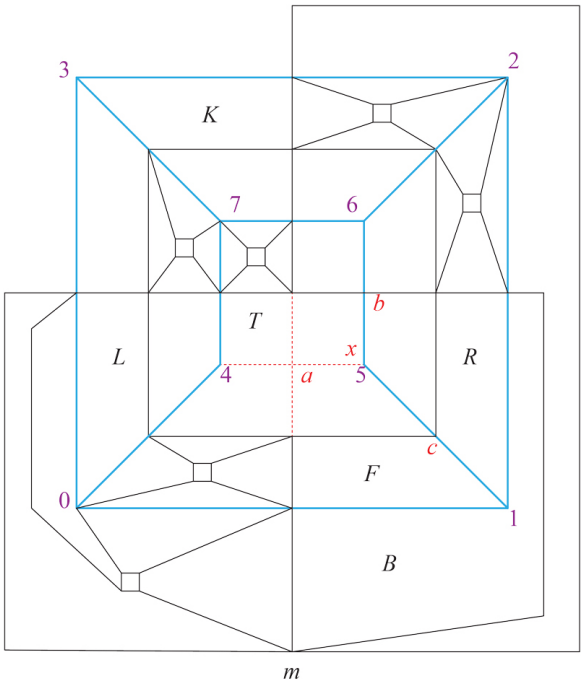


Figure 7: Schlegel diagram of G_{-a}. We follow [DF18] in labeling the faces of a cube as F, K, R, L, T, B for Front, bacK, Right, Left, Top, Bottom respectively. The corners of P_{222} are labeled $0, 1, 2, 3$ around the bottom face B, and $4, 5, 6, 7$ around the top face T. m is the vertex in the middle of B. The edges deleted by removing vertex a are shown dashed.

of P_{14} is 2 extra 0's. Deleting a maintains bipartiteness and increases the parity imbalance of G_{-a} to 3. Therefore by Lemma 3, G_{-a} has no Hamiltonian path, and such a ρ cannot exist.

(2) **ρ includes just one flat vertex c, and excludes a, b.** (Again symmetry ensures there is no loss of generality in assuming the one included flat vertex is c.) ρ must include corner x, which is only accessible in $G_{P_{14}}$ through the three flat vertices. If ρ excludes a, b, then it must include the edge cx. Let $G_{-ab} = G_{P_{14}} \setminus \{a, b\}$. In G_{-ab}, x has degree 1, so ρ terminates there. It must be that ρ is a Hamiltonian path in G_{-ab}, but the deletion of a, b increases the parity imbalance to 4, and so again such a Hamiltonian path cannot exist.

(3) **ρ excludes a, b, c.** Because corner x is only accessible through one of these flat vertices, ρ never reaches x and so cannot be an edge zipper unfolding.

Thus the assumption that there is an edge zipper unfolding cut path ρ for P_{14} reaches a contradiction in all four cases. Therefore, there is no edge zipper unfolding cut path for P_{14}.[2] □

5 Edge Unfoldings of P_{14} and P_{44}

Now that it is known that P_{14} and P_{44} each have no edge zipper unfolding, it is natural to wonder whether either settles the edge-unfolding open problem: can they be edge unfolded? Indeed both can: see Figures 8 and 9.

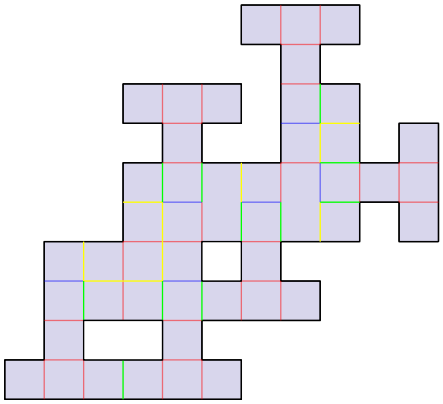


Figure 8: Edge unfolding of P_{14}. Colors: green = cut, red = mountain, blue = valley, yellow = flat.

The colors in these layouts are those used by Origami Simulator [GDG18]. Fig. 10 shows a partial folding

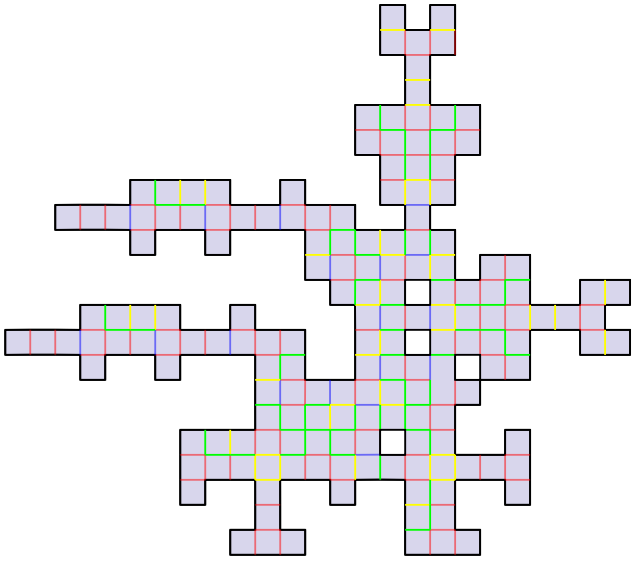


Figure 9: Edge unfolding of P_{44}. Colors: green = cut, red = mountain, blue = valley, yellow = flat.

of P_{44}, and animations are at `http://cs.smith.edu/~jorourke/Unf/NoEdgeUnzip.html`.

[2] Just to verify this conclusion, we constructed these graphs in Mathematica and `FindHamiltonianPath[]` returned {} for each.

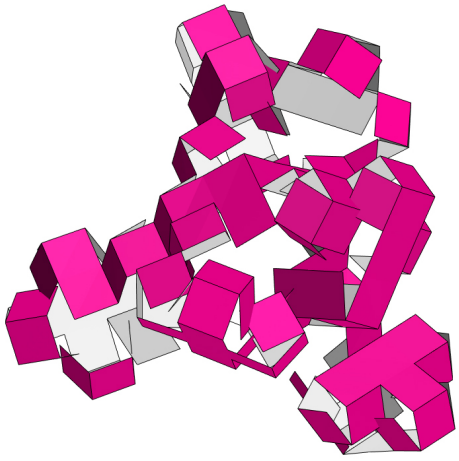


Figure 10: Partial folding of the layout in Fig. 9. Compare with Fig. 4.

6 Many Polycubes with No Edge Zipper Unfolding

As pointed out by Ryuhei Uehara,[3] P_{44} can be extended to an infinite number of polycubes with no zipper unfolding. Let P'_6 be the polycube in Fig. 2 with the bottom cube removed. So P'_6 has a '+' sign of five cubes in its base layer. Let B be the bottom face of the cube at the center of the '+' sign. Attach P'_6 to the highest cube of P_{44} in Fig. 1(a) by gluing B to the top face of that top cube. It is easy to verify that all new vertices of this augmented object, call it P'_{44}, are corners. The joining process can be repeated with another copy of P'_6, producing P''_{44}, and so on. All of these polycubes have no zipper unfolding.

We have not attempted to edge-unfold these larger objects.

7 Open Problems

The most interesting question remaining in this line of investigation is **Question 1** (Sec. 2): Does every polycube tree have an edge zipper unfolding?

Acknowledgements. We thank participants of the Bellairs 2018 workshop for their insights. We benefitted from suggestions by the referees.

References

[BDD+98] Therese Biedl, Erik D. Demaine, Martin L. Demaine, Anna Lubiw, Joseph O'Rourke, Mark Overmars, Steve Robbins, and Sue Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proc. 10th Canad. Conf. Comput. Geom.*, pages 70–71, 1998. Full version in *Elec. Proc.*: `http://cgm.cs.mcgill.ca/cccg98/proceedings/cccg98-biedl-unfolding.ps.gz`.

[DDL+10] Erik D. Demaine, Martin L. Demaine, Anna Lubiw, Arlo Shallit, and Jonah Shallit. Zipper unfoldings of polyhedral complexes. In *Proc. 22nd Canad. Conf. Comput. Geom.*, pages 219–222, August 2010.

[DDU13] Erik D. Demaine, Martin L. Demaine, and Ryuhei Uehara. Zipper unfoldability of domes and prismoids. In *Proc. 25th Canad. Conf. Comput. Geom.*, August 2013.

[DF18] Mirela Damian and Robin Flatland. Unfolding orthotrees with constant refinement. `http://arxiv.org/abs/1811.01842`, 2018.

[GDG18] Amanda Ghassaei, Erik D. Demaine, and Neil Gershenfeld. Fast, interactive origami simulation using GPU computation. In *Origami 7: 7th Internat. Mtg. Origami Science, Mathematics and Education (OSME)*, 2018.

[O'R10] Joseph O'Rourke. Flat zipper-unfolding pairs for Platonic solids. `http://arxiv.org/abs/1010.2450`, October 2010.

[O'R19] Joseph O'Rourke. Unfolding polyhedra. In *Proc. 31st Canad. Conf. Comput. Geom.*, August 2019.

[She75] Geoffrey C. Shephard. Convex polytopes with convex nets. *Math. Proc. Camb. Phil. Soc.*, 78:389–403, 1975.

[3] Personal communication, June 2020.

Acutely Triangulated, Stacked, and Very Ununfoldable Polyhedra

Erik D. Demaine* Martin L. Demaine* David Eppstein†

Abstract

We present new examples of topologically convex edge-ununfoldable polyhedra, i.e., polyhedra that are combinatorially equivalent to convex polyhedra, yet cannot be cut along their edges and unfolded into one planar piece without overlap. One family of examples is ***acutely triangulated***, i.e., every face is an acute triangle. Another family of examples is ***stacked***, i.e., the result of face-to-face gluings of tetrahedra. Both families achieve another natural property, which we call ***very ununfoldable***: for every k, there is an example such that every nonoverlapping multipiece edge unfolding has at least k pieces.

1 Introduction

Can every convex polyhedron be cut along its edges and unfolded into a single planar piece without overlap? Such ***edge unfoldings*** or ***nets*** are useful for constructing 3D models of a polyhedron (from paper or other material such as sheet metal): cut out the net, fold along the polyhedron's uncut edges, and re-attach the polyhedron's cut edges [25]. Unfoldings have also proved useful in computational geometry algorithms for finding shortest paths on the surface of polyhedra [3, 5, 9].

Edge unfoldings were first described in the early 16th century by Albrecht Dürer [16], implicitly raising the still-open question of whether every convex polyhedron has one (sometimes called Dürer's conjecture). The question was first formally stated in 1975 by G. C. Shephard, although without reference to Dürer [17, 23]. It has been heavily studied since then, with progress of two types [15, 22]:

1. finding restricted classes of polyhedra, or generalized types of unfoldings, for which the existence of an unfolding can be guaranteed; and

2. finding generalized classes of polyhedra, or restricted types of unfoldings, for which counterexamples — ***ununfoldable polyhedra*** — can be shown to exist.

*Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, {edemaine,mdemaine}@mit.edu

†Computer Science Department, University of California, Irvine, eppstein@uci.edu. This work was supported in part by the US National Science Foundation under grant CCF-1616248.

Results guaranteeing the existence of an unfolding include:

- Every pyramid, prism, prismoid, and dome has an edge unfolding [15].

- Every sufficiently flat acutely triangulated convex terrain has an edge unfolding [21]. Consequentially, every acutely triangulated convex polyhedron can be unfolded into a number of planar pieces that is bounded in terms of the "acuteness gap" of the polyhedron, the minimum distance of its angles from a right angle.

- Every convex polyhedron has an affine transformation that admits an edge unfolding [18].

- Every convex polyhedron can be unfolded to a single planar piece by cuts interior to its faces [3, 14].

- Every polyhedron with axis-parallel sides can be unfolded after a linear number of axis-parallel cuts through its faces [10].

- Every triangulated surface (regardless of genus) has a "vertex unfolding", a planar layout of triangles connected through their vertices that can be folded into the given surface [13].

- For ideal polyhedra in hyperbolic space, unlike Euclidean convex polyhedra or non-ideal hyperbolic polyhedra, every spanning tree forms the system of cuts of a convex unfolding into the hyperbolic plane.

Previous constructions of ununfoldable polyhedra include the following results. A polyhedron is ***topologically convex*** if it is combinatorially equivalent to a convex polyhedron, meaning that its surface is a topological sphere and its graph is a 3-vertex-connected planar graph.

- Some orthogonal polyhedra and topologically convex orthogonal polyhedra have no edge unfolding, and it is NP-complete to determine whether an edge unfolding exists in this case [1, 8].

- There exists a convex-face star-shaped topologically convex polyhedron with no edge unfolding [20, 24].

- There exists a triangular-face topologically convex polyhedron with no edge unfolding [7].
- There exist edge-ununfoldable topologically convex polyhedra with as few as 7 vertices and 6 faces, or 6 vertices and 7 faces [4].
- There exists a topologically convex polyhedron that does not even have a vertex unfolding [2].
- There exist domes that have no ***Hamiltonian unfolding***, in which the cuts form a Hamiltonian path through the graph of the polyhedron [12]. Similarly, there exist polycubes that have no Hamiltonian unfolding [11].
- There exists a convex polyhedron, equipped with 3-vertex-connected planar graph of geodesics partitioning the surface into regions metrically equivalent to convex polygons, that cannot be cut and unfolded along graph edges [6].

In this paper, we consider two questions left open by the previous work on edge-ununfoldable polyhedra with triangular faces [7], and strongly motivated by O'Rourke's recent results on unfoldings of acutely-triangulated polyhedra [21]. First, the previous counterexample of this type involved triangles with highly obtuse angles. Is this a necessary feature of the construction, or does there exist an ununfoldable polyhedron with triangular faces that are all acute? Second, how far from being unfoldable can these examples be? Is it possible to cut the surfaces of these polyhedra into a bounded number of planar pieces (instead of a single piece) that can be folded and glued to form the polyhedral surface? (Both questions are motivated by previously posed analogous questions for convex polyhedra, as easier versions of Dürer's conjecture [15, Open Problems 22.12 and 22.17].)

We answer both of these questions negatively, by finding families of topologically convex edge-ununfoldable polyhedra with all faces acute triangles, in which any cutting of the surface into regions that can be unfolded to planar pieces must use an arbitrarily large number of pieces. Additionally, we use a similar construction to prove that there exist edge-ununfoldable ***stacked polyhedra*** [19], formed by gluing tetrahedra face-to-face with the gluing pattern of a tree, that also require an arbitrarily large number of pieces to unfold. We leave open the question of whether there exists an edge-ununfoldable stacked polyhedron with acute-triangle faces.

2 Hats

Our construction follows that of Bern et al. [7] in being based on certain triangulated topological disks, which they called ***hats***. The combinatorial structure of a hat (in top view, but with different face angles than the hat we use in our proof) is shown in Figure 1: It consists of nine triangles, three of which (the ***brim***, blue in the figure) have one edge on the outer boundary of the disk. The next three triangles, yellow in the figure, have a vertex but not an edge on the disk boundary; we call these the ***band*** of the hat. The central three triangles, pink in the figure, are disjoint from the boundary and meet at a central vertex; we call these the ***crown*** of the hat.

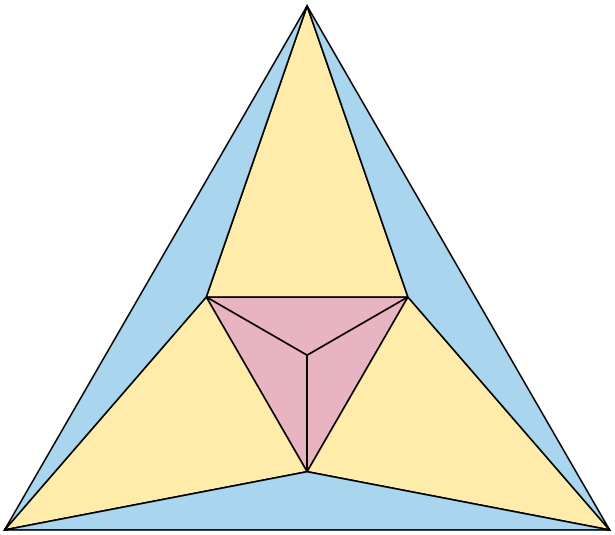


Figure 1: Combinatorial structure of a hat

In both the construction of Bern et al. [7] and in our construction, the three vertices of the hat that are interior to the disk but not at the center all have negative curvature, meaning that the sum of the angles of the faces meeting at these vertices is greater than 2π. The center vertex, on the other hand, has positive curvature, a sum of angles less than 2π. When this happens, we can apply the following lemmas:

Lemma 2.1 *At any negatively-curved vertex of a polyhedron, any unfolding of the polyhedron that cuts only its edges and separates its surface into one or more simple polygons must cut at least two edges at each negatively-curved vertex.*

Proof. If only one edge were cut then the faces surrounding that vertex could not unfold into the plane without overlap. □

Lemma 2.2 *Let D be a subset of the faces of a polyhedron, such that the polyhedron is topologically a sphere and D is topologically equivalent to a disk (such as a hat). Then in any unfolding of the polyhedron (possible cutting it into multiple pieces), either D is separated*

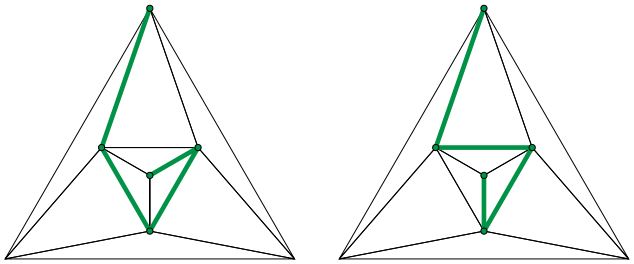


Figure 2: Two paths from a boundary vertex of a hat, through all three negatively curved vertices, to the center vertex

into multiple pieces by a path of cut edges from one boundary vertex of D to another or by a cycle of cut edges within D, or the set of cut edges within D forms a forest with at most one boundary vertex for each tree in the forest.

Proof. If the cut edges within D do not form a forest, they contain a cycle and the Jordan Curve Theorem implies that this cycle separates an interior part of the boundary from the exterior. If they form a forest in which some tree contains two boundary vertices, then they contain a boundary-to-boundary path within D, again separating D by the Jordan Curve Theorem. The only remaining possibility is a forest with at most one boundary vertex per tree. □

Lemma 2.3 *For a hat combinatorially equivalent to the one in Figure 1, with positive curvature at the center vertex and negative curvature at the other three interior vertices, any unfolding that does not cut the hat into multiple pieces must cut a set of edges along a single path from a boundary vertex to the center vertex.*

Proof. By Lemma 2.2, each component of cut edges must form a tree with at most one boundary vertex within the hat. But every tree with one or more edges has at least two leaves, and every tree that is not a path has at least three leaves. By Lemma 2.1, the only non-boundary leaf can be the center vertex, so each component must be a path from the boundary to this vertex. □

Up to symmetries of the hat, there are only two distinct shapes that the path of Lemma 2.3 from the boundary to the center of a hat can have (Figure 2). These two cuttings differ in how the crown triangles are attached to the band and to each other, but they both cut the brim and band triangles in the same way, into a strip of triangles connected edge-to-edge around the boundary of the hat.

Our key new construction is depicted in unfolded (but self-overlapping) form in Figure 3. It is a hat in which all triangles are acute and isosceles:

- The three brim triangles have apex angle 85° and base angle 47.5°.
- The three band triangles have base angle 85° and apex angle 10°.
- The three crown triangles are congruent to the band triangles, with base angle 85° and apex angle 10°.

As in the construction of Bern et al. [7], this leaves negative curvature (total angle 425° from five 85° angles) at the three non-central interior angles of the hat, and positive curvature (total angle 30°) at the center vertex, allowing the lemmas above to apply. The cut edges of the figure form a tree with a degree-three vertex at one of the negatively curved vertices of the hat, and a leaf at another negatively curved vertex, the one at which the self-overlap of the figure occurs, So the cutting in the figure does not match in detail either of the two path cuttings of Figure 2. Nevertheless, the brim and band triangles are unfolded as they would be for either of these two path cuttings. It is evident from the figure that this unfolding of the brim and band triangles cannot be extended to a one-piece unfolding of the entire hat: if a crown triangle is attached to the middle of the three unfolded band triangles (as it is in the figure) then there is no room on either side of it to attach the other two crown triangles, and a crown triangle attached to either of the other two band triangles would overlap the opposite band triangle. We prove this visual observation more formally below.

Lemma 2.4 *The hat with acute triangles described above has no single-piece unfolding.*

Proof. As we have already seen in Lemma 2.3, any unfolding (if it exists) must be along one of the two cut paths depicted in Figure 2. As a result, the unfolding of the brim and band triangles (but not the crown triangles) must be as depicted in Figure 3. In this unfolding, the three base sides of the unfolded band triangles form a polygonal chain whose interior angles (surrounding the central region of the figure where the pink crown triangles are attached) can be calculated as 105°.

A regular pentagon has interior angles of 108°, and has the property that each vertex lies on the perpendicular bisector of the opposite edge. Because the interior angles of the chain of base sides of band triangles are 105°, less than this 108° angle, it follows that the band triangle at one end of the chain extends across the perpendicular bisector of the base edge at the other end of the chain. Further, it does so at a point closer than the vertex of a regular pentagon sharing this same base edge (Figure 4).

If a crown triangle were attached to one of the two base edges at the ends of the chain of three base edges,

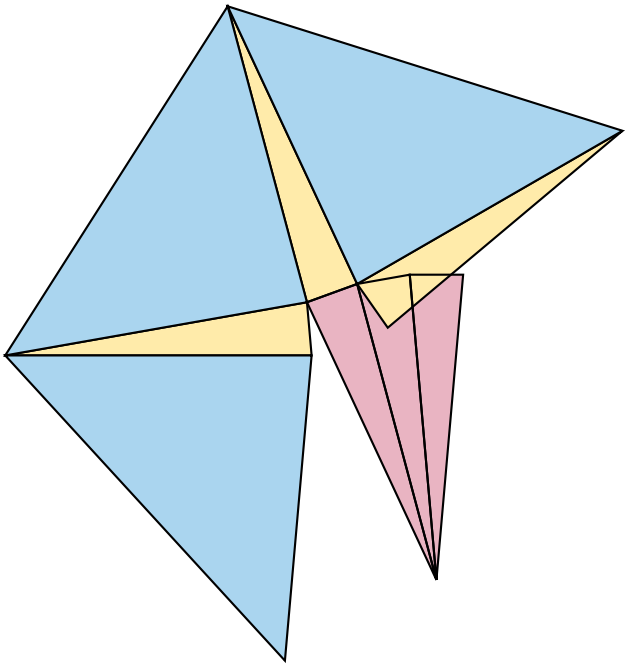


Figure 3: A hat made with acute isosceles triangles. Unlike Figure 2, the cuts made to form the self-overlapping unfolding shown do not form a path.

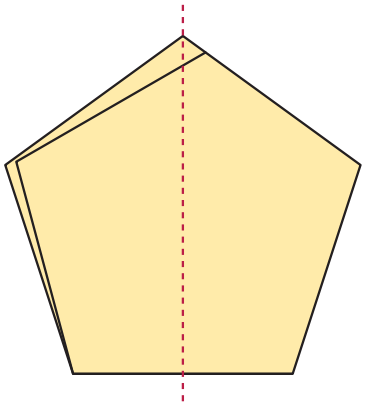


Figure 4: Each vertex of a regular pentagon lies on the perpendicular bisector of the opposite side; in a path of three equal edges with the tighter angle 105°, the last edge overlaps the perpendicular bisector of the first.

its altitude would lie along the perpendicular bisector of the base edge. And because the crown triangle has an apex angle of 10°, sharper than the angle of an isosceles triangle inscribed within a regular pentagon, its altitude extends across the perpendicular bisector farther than the regular pentagon vertex, causing it to overlap with the band triangle at the other end of the chain of three base edges.

Therefore, attaching a crown triangle to either the first or last of the band triangle base edges in the chain of these three edges necessarily leads to a self-overlapping unfolding. However, these two ways of attaching a crown triangle are the only ones permitted by the two cases depicted in Figure 2. Attaching a crown triangle to the middle of the three base edges, as in Figure 3, can only be done by cutting along a tree that is not a path. Therefore, no unfolding exists. □

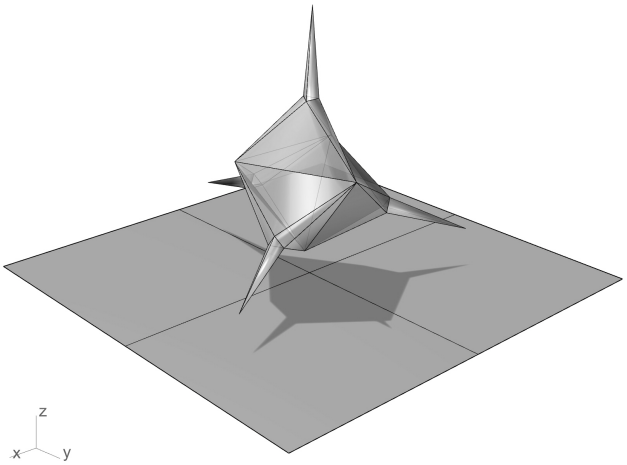


Figure 5: Tetrahedron with faces replaced by hats

The following construction is straightforward, and will allow us to construct polyhedra with multiple hats while keeping the hats disjoint from each other.

Lemma 2.5 *The hat with acute triangles described above can be realized in three-dimensional space, lying within a right equilateral-triangle prism whose base is the boundary of the hat.*

3 Acute Ununfoldable Polyhedra

We now use these hats to construct a topologically convex ununfoldable polyhedron.

Theorem 3.1 *There exists a topologically convex ununfoldable polyhedron whose faces are all acute isosceles triangles.*

Proof. Replace the four faces of a regular tetrahedron by acute-triangle hats, all pointing outward, as shown in Figure 5. Because each lies within a prism having the tetrahedron face as a base, they do not overlap each other in space. By Lemma 2.4, no hat can be unfolded into a single piece, so any possible unfolding (even one into multiple pieces) must cut each hat along some path between two of its three boundary vertices (at least; there may be more cuts besides these). The four paths

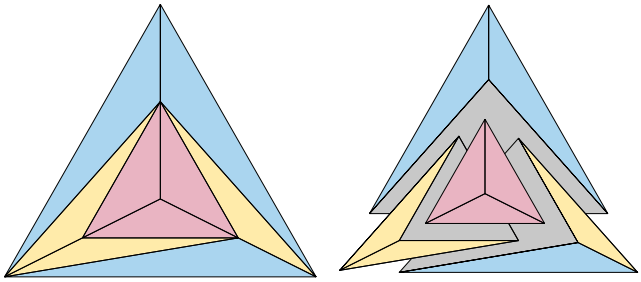


Figure 6: Hat for stacked polyhedra (top view, left, and exploded view as a stacked polyhedron, right)

formed in this way are disjoint except at their ends, and connect the four vertices of the tetrahedron, necessarily forming at least one cycle that separates the tetrahedron into at least two pieces. □

Like the examples of Tarasov, Grünbaum, and Bern et al. [7, 20, 24], the resulting polyhedron is also star-shaped, with the center of the tetrahedron in its kernel.

4 Stacked Ununfoldable Polyhedra

A ***stacked polyhedron*** is a polyhedron that can be formed by repeatedly gluing a tetrahedron onto a single triangular face of a simpler stacked polyhedron, starting from a single tetrahedron [19]. To make ununfoldable stacked polyhedra, we use a similar strategy to our construction of ununfoldable polyhedra with acute-triangle-faces, in which we replace some faces of a convex polyhedron by hats. However, the acute-triangle hat that we used earlier cannot be used as part of a stacked polyhedron: in a stacked polyhedron, every non-face triangle is subdivided into three smaller triangles, but that is not true of the outer triangle of Figure 1. Instead, we use the hat shown in Figure 6. As before, it has three brim triangles, three band triangles, and three crown triangles, but they are arranged differently and less symmetrically. We make the brim and band triangles nearly coplanar, with shapes approximating those shown in the figure, but projecting slightly out of the figure so that the result can be constructed as a stacked polyhedron. We choose the crown triangles to be isosceles, and taller than the isosceles triangles inscribed in regular pentagons, as in our acute-triangle construction, so that (as viewed in Figure 6) they project out of the figure.

Lemma 4.1 *The hat described above has no single-piece unfolding.*

Proof. As with our other hat, the center vertex of this hat has positive curvature, and the other three interior vertices have negative curvature, so by Lemma 2.3 any unfolding of the hat that leaves it in one piece must form a path consisting of a single edge cutting from the boundary to the crown, two edges cutting between the band and the crown, and one edge cutting to the center of the crown.

There are many more cases than there were in Figure 2, but we can avoid case-based reasoning by arguing that in each case, the brim and band triangles unfold in such a way that the three edges between the band and crown triangles form a polygonal chain with interior angles less than the 108° angles of the regular pentagon (in fact, close to 60°, because of the way we have constructed this part of the hat to differ only by a small amount from the top view shown in Figure 6. Therefore, just as in Figure 4, each edge at one end of this chain of three edges overlaps the perpendicular bisector of the edge at the other end of the chain.

Cutting along a path from a boundary edge of the hat to its center vertex forces the three crown triangles to be attached to the unfolded brim and band triangles on one of the two edges at the end of this path. However, our construction makes the three crown triangles tall enough to ensure that, no matter which of these two edges they are attached to, they will overlap the edge at the other end of the path at the point where it crosses the perpendicular bisector. □

Theorem 4.2 *There exists an ununfoldable stacked polyhedron.*

Proof. We replace the four faces of a regular tetrahedron with the hat described above. Each such replacement can be realized as a stacking of four tetrahedra onto the face, so the result is a stacked polyhedron. As in Theorem 3.1, each hat lies within a prism having the tetrahedron face as a base, so they do not overlap each other in space; and the set of edges cut in any unfolding must include at least four paths between the four tetrahedron vertices, necessarily forming a cycle that cuts one part of the polyhedron surface from the rest. □

A stacked hat with the same combinatorial structure as the one used in this construction, with the center vertex positively curved and the surrounding three vertices negatively curved, cannot be formed from acute triangles, because that would leave the degree-four vertex with positive curvature. We leave as an open question whether it is possible for an ununfoldable stacked polyhedron to have all faces acute.

5 Very Ununfoldable Polyhedra

Both families of examples above can be made into very ununfoldable families. In both cases, the approach is the same: instead of starting from a tetrahedron, we start from a polyhedron with many triangular faces, and show that attaching hats to more and more triangles requires more and more unfolded pieces.

Theorem 5.1 *There exist topologically convex polyhedra with acute isosceles triangle faces such that any unfolding formed by cutting along edges into multiple non-self-overlapping pieces requires an unbounded number of pieces.*

Proof. For any integer $k \geq 1$, refine the regular tetrahedron by subdividing each edge into k equal-length edges and subdivide each face into a regular grid of $\sum_{i=1}^{k}(2i-1) = k^2$ equilateral triangles of side length $1/k$, for a total of $4k^2$ faces and (by inclusion-exclusion) $\sum_{i=1}^{k+1} i - 6(k+1) + 4 = 2k^2 + 2$ vertices. Replace each equilateral triangular face by an acute-triangle hat pointing outward. As in Theorem 3.1, each hat lies within a prism having the face of the tetrahedron as a base, so they do not overlap each other in space; and any unfolding into multiple pieces must, in each hat, either cut along a cycle within the hat or cut along some path connecting two of its three boundary vertices. Let c be the number of cycles within hats cut in this way, so that we have a system of at least $4k^2 + c$ disjoint paths connecting pairs of subdivided-tetrahedron vertices.

Now consider cutting the polyhedron surface along these paths, one by one. Each cut either connects two subdivided-tetrahedron vertices that were not previously connected along the system of cuts, or two subdivided-tetrahedron vertices that were previously connected. If cutting along a path connects two vertices that were not previously connected, it reduces the number of connected components among these vertices; this case can happen at most $2k^2 + 1$ times. If cutting along a path connects two vertices that were previously connected, then that path and the path through which they were previously connected form a Jordan curve that separates off two parts of the surface from each other. Because there are $4k^2 - c$ paths connecting pairs of subdivided-tetrahedron vertices, only $2k^2 + 1$ of which can form new connections, this case must happen at least $2k^2 - 1 - c$ times. Because the surface started with a single piece and undergoes at least $2k^2 - 1 - c$ separations, it ends up with at least $2k^2 - c$ pieces, which together with the c additional pieces formed by cycles within hats, form a total of at least $2k^2$ pieces. □

Theorem 5.2 *There exist topologically convex stacked polyhedra such that any unfolding formed by cutting along edges into multiple non-self-overlapping pieces requires an unbounded number of pieces.*

Proof. For any integer $k \geq 0$, refine the regular tetrahedron by choosing any face and attaching to the face a very shallow tetrahedron whose apex is near the in-center of the face, effectively splitting the face into three faces, and repeating this process a total of k times. Because each attachment increases the number of faces by 2 and the number of vertices by 1, the result is a stacked polyhedron with $4 + 2k$ triangular faces (not necessarily equilateral) and $4 + k$ vertices. Replace each triangle with a version of the hat from Section 4 pointed outward, using the availability flexibility to make the interface between the band and crown an equilateral triangle near the in-center of the original triangle. As in Theorem 4.2, the result is a stacked polyhedron; each hat lies within a prism having the face of the tetrahedron as a base, so they do not overlap each other in space; and any unfolding into multiple pieces must cut each hat along some path connecting two of its three boundary vertices. As in Theorem 5.1, at most $3 + k$ such paths can decrease the number of connected components among the $4 + k$ vertices, leaving at least $1 + k$ paths that separate the surface into at least $2 + k$ pieces. □

Acknowledgments

This research was initiated during the Virtual Workshop on Computational Geometry organized by E. Demaine on March 20–27, 2020. We thank the other participants of that workshop for helpful discussions and providing an inspiring atmosphere.

References

[1] Zachary Abel and Erik D. Demaine. Edge-Unfolding Orthogonal Polyhedra is Strongly NP-Complete. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG 2011)*, 2011. URL: https://www.cccg.ca/proceedings/2011/papers/paper43.pdf.

[2] Zachary Abel, Erik D. Demaine, and Martin L. Demaine. A topologically convex vertex-ununfoldable polyhedron. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG 2011), Toronto, August 10–12, 2011*, 2011. URL: https://cccg.ca/proceedings/2011/papers/paper85.pdf.

[3] Pankaj K. Agarwal, Boris Aronov, Joseph O'Rourke, and Catherine A. Schevon. Star unfolding of a polytope with applications. *SIAM Journal on Computing*, 26(6):1689–1713, 1997. doi:10.1137/S0097539793253371.

[4] Hugo A. Akitaya, Erik D. Demaine, David Eppstein, Tomohiro Tachi, and Ryuhei Uehara. Minimal ununfoldable polyhedron. In *Abstracts from the 22nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2019)*, pages 27–28, Tokyo, Japan, September 2019. URL: https://erikdemaine.org/papers/MinimalUnunfoldable_JCDCGGG2019/.

[5] Boris Aronov and Joseph O'Rourke. Nonoverlap of the star unfolding. *Discrete & Computational Geometry*, 8(3):219–250, 1992. doi:10.1007/BF02293047.

[6] Nicholas Barvinok and Mohammad Ghomi. Pseudo-edge unfoldings of convex polyhedra. *Discrete & Computational Geometry*, 2019. doi:10.1007/s00454-019-00082-1.

[7] Marshall Bern, Erik D. Demaine, David Eppstein, Eric Kuo, Andrea Mantler, and Jack Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry: Theory & Applications*, 24(2):51–62, 2003. doi:10.1016/S0925-7721(02)00091-3.

[8] Therese C. Biedl, Erik D. Demaine, Martin L. Demaine, Anna Lubiw, Mark H. Overmars, Joseph O'Rourke, Steve Robbins, and Sue Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proceedings of the 10th Canadian Conference on Computational Geometry (CCCG 1998)*, 1998. URL: https://cgm.cs.mcgill.ca/cccg98/proceedings/cccg98-biedl-unfolding.ps.gz.

[9] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *Proceedings of the 6th Annual Symposium on Computational Geometry (SoCG 1990)*. ACM Press, 1990. doi:10.1145/98524.98601.

[10] Mirela Damian, Erik D. Demaine, Robin Flatland, and Joseph O'Rourke. Unfolding genus-2 orthogonal polyhedra with linear refinement. *Graphs and Combinatorics*, 33(5):1357–1379, 2017. doi:10.1007/s00373-017-1849-5.

[11] Erik D. Demaine, Martin L. Demaine, David Eppstein, and Joseph O'Rourke. Some polycubes have no edge-unzipping. Electronic preprint arxiv:1907.08433, 2019.

[12] Erik D. Demaine, Martin L. Demaine, and Ryuhei Uehara. Zipper unfoldability of domes and prismoids. In *Proceedings of the 25th Canadian Conference on Computational Geometry (CCCG 2013), Waterloo, Ontario, Canada August 8th–10th, 2013*, 2013. URL: https://cccg.ca/proceedings/2013/papers/paper_10.pdf.

[13] Erik D. Demaine, David Eppstein, Jeff Erickson, George W. Hart, and Joseph O'Rourke. Vertex-unfoldings of simplicial manifolds. In *Discrete Geometry: In honor of W. Kuperberg's 60th birthday*, volume 253 of *Pure and Applied Mathematics*, pages 215–228. Marcel Dekker, 2003.

[14] Erik D. Demaine and Anna Lubiw. A generalization of the source unfolding of convex polyhedra. In Alberto Márquez, Pedro Ramos, and Jorge Urrutia, editors, *Revised Papers from the 14th Spanish Meeting on Computational Geometry*, volume 7579 of *Lecture Notes in Computer Science*, pages 185–199, Alcalá de Henares, Spain, June 2011. doi:10.1007/978-3-642-34191-5_18.

[15] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007.

[16] Albrecht Dürer. *The Painter's Manual: A Manual of Measurement of Lines, Areas, and Solids by Means of Compass and Ruler Assembled by Albrecht Dürer for the Use of All Lovers of Art with Appropriate Illustrations Arranged to be Printed in the Year MDXXV*. Abaris Books, Inc., New York, 1977. English translation of *Unterweysung der Messung mit dem Zirkel un Richtscheyt in Linien Ebnen und Gantzen Corporen*, 1525.

[17] Michael Friedman. *A History of Folding in Mathematics: Mathematizing the Margins*. Birkhäuser, 2018. See in particular page 47. doi:10.1007/978-3-319-72487-4.

[18] Mohammad Ghomi. Affine unfoldings of convex polyhedra. *Geometry & Topology*, 18:3055–3090, 2014. doi:10.2140/gt.2014.18.3055.

[19] Branko Grünbaum. A convex polyhedron which is not equifacettable. *Geombinatorics*, 10(4):165–171, 2001. URL: https://sites.math.washington.edu/~grunbaum/Nonequifacettablesphere.pdf.

[20] Branko Grünbaum. No-net polyhedra. *Geombinatorics*, 11:111–114, 2002. URL: https://www.math.washington.edu/~grunbaum/Nonetpolyhedra.pdf.

[21] Joseph O'Rourke. Edge-unfolding nearly flat convex caps. In Bettina Speckmann and Csaba D. Tóth, editors, *Proceedings of the 34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *LIPIcs*, pages 64:1–64:14, 2018. doi:10.4230/LIPIcs.SoCG.2018.64.

[22] Joseph O'Rourke. Unfolding polyhedra. Electronic preprint arxiv:1908.07152, 2019.

[23] G. C. Shephard. Convex polytopes with convex nets. *Mathematical Proceedings of the Cambridge Philosophical Society*, 78(3):389–403, 1975. doi:10.1017/s0305004100051860.

[24] A. S. Tarasov. Polyhedra that do not admit natural unfoldings. *Uspekhi Matematicheskikh Nauk*, 54(3):185–186, 1999. doi:10.1070/rm1999v054n03ABEH000171.

[25] Magnus J. Wenninger. *Polyhedron Models*. Cambridge University Press, 1971.

Nets of higher-dimensional cubes

Kristin DeSplinter* Satyan L. Devadoss† Jordan Readyhough‡ Bryce Wimberly§

Abstract

In this extended abstract, we show that every ridge unfolding of an n-cube is without self-overlap, yielding a valid net. The results are obtained by developing machinery that translates cube unfolding into combinatorial frameworks. The bounding boxes of these cube nets are also explored using integer partitions.

1 Introduction

The study of unfolding polyhedra was popularized by Albrecht Dürer in the early 16th century in his influential book *The Painter's Manual*. It contains the first recorded examples of polyhedral *nets*, connected edge unfoldings of polyhedra that lay flat on the plane without overlap. Motivated by this, Shephard [6] conjectures that every convex polyhedron can be cut along certain edges and admits a net. This claim remains tantalizingly open.

We consider this question for higher-dimensional convex *polytopes*. The codimension-one faces of a polytope are *facets* and its codimension-two faces are *ridges*. The analog of an edge unfolding of polyhedron is the *ridge unfolding* of an n-dimensional polytope: the process of cutting the polytope along a collection of its ridges so that the resulting (connected) arrangement of its facets develops isometrically into an $\mathbb{R}^{n-1}$ hyperplane.

There is a rich history of higher-dimensional unfoldings of polytopes, with the collected works of Alexandrov [1] serving as seminal reading. In 1984, Turney [7] enumerates the 261 ridge unfoldings of the 4-cube, and in 1998, Buekenhout and Parker [2] extend this enumeration to the other five regular convex 4-polytopes. Both of these works focus on enumerative rather than geometric unfolding results. Miller and Pak [4] construct an algorithm which provides an unfolding of polytopes without overlap. However, their method allows cuts interior to facets, not just along ridges.

Our work targets ridge unfoldings of the n-cube. For the 3-cube, Figure 1 shows the 11 different unfoldings (up to symmetry), all of which yield nets. Section 2 generalizes this into our main result: *every* ridge unfolding of the n-cube results in a net. Section 3 considers packing these cube nets into boxes using integer partitions. Finally, we form a conjecture concerning regular convex polytopes in Section 4.

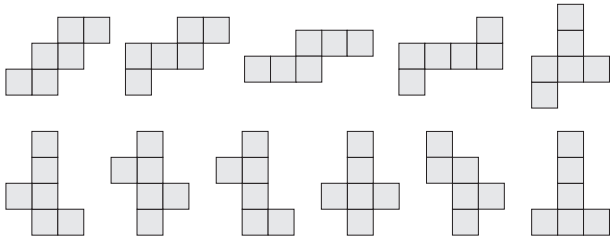


Figure 1: The 11 edge unfoldings of the 3-cube.

2 Rolling and Unfolding

We explore ridge unfoldings of a convex polytope P by focusing on the combinatorics of the arrangement of its facets in the unfolding. In particular, a ridge unfolding induces a tree whose nodes are the facets of the polytope and whose edges are the uncut ridges between the facets [5]. Indeed, this is a spanning tree in the 1-skeleton of the dual of P.

The dual of the n-cube is the n-orthoplex, whose 1-skeleton forms the *n-Roberts graph*. The $2n$ nodes of this graph (corresponding to the $2n$ facets of the n-cube) can arranged on a circle so that antipodal nodes represent opposite facets of the cube. Thus, unfoldings of an n-cube are in bijection with spanning trees of the n-Roberts graph.

Example 1 *Figure 2(a) considers an edge unfolding of the 3-cube with its underlying dual tree. This appears as a spanning tree on the 1-skeleton of the octahedral dual (b), redrawn on the* 3*-Roberts graph (c).*

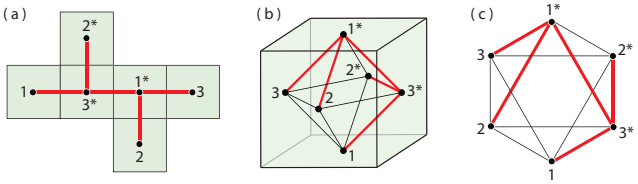


Figure 2: An unfolding of a 3-cube with its corresponding spanning tree on the 3-Roberts graph.

*Department of Mathematics, University of Utah, desplinter.k@utah.edu
†Department of Mathematics, University of San Diego, devadoss@sandiego.edu
‡School of Architecture, Columbia University, jhr2150@columbia.edu
§Trident Seafood Analysis, bwimberly2@gmail.com

Recall that a ridge unfolding of an n-cube is a connected arrangement of its $2n$ facets, developed isometrically into hyperplane $\mathbb{R}^{n-1}$. Begin the unfolding by choosing a (base) facet b of the n-cube, placing it on the hyperplane. Then the normal vector n_b to b becomes normal to the hyperplane. Consider an adjacent facet c to b, and roll the cube along the ridge between these facets, with facet c now landing on the hyperplane. Figure 3 shows a rendering of the orthogonal projection of such a roll, with $c*$ and $b*$ corresponding to the antipodal facets of c and b, and the marked red edge representing the ridge between c and b.

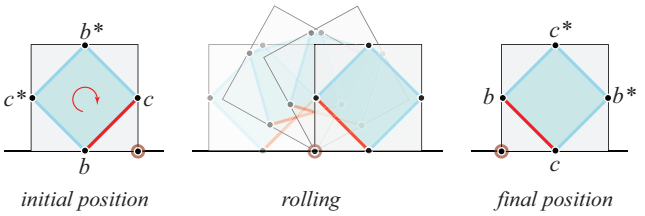


Figure 3: Rolling a cube on a hyperplane.

Since we rotate only along the plane spanned by the normal vectors n_b and n_c, the remaining directions stay fixed in the development. This is captured combinatorially as a rotation of a subgraph of the Roberts graph:

Definition 1 *A* roll *from base facet b towards an adjacent facet c rotates the four nodes $\{b, c, b*, c*\}$ of the Roberts graph along the quadrilateral (keeping the remaining nodes fixed), making c the new base facet.*

Figure 4 shows an example for the 5-cube, where the highlighted quadrilateral (depicting the roll) is invoking the colored square of Figure 3.

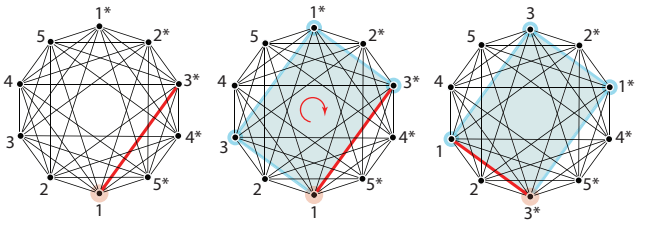


Figure 4: Rotating facet 1 towards 3^* on a 5-cube.

The advantage of unfolding a cube (compared to an arbitrary convex polytope) into hyperplane $\mathbb{R}^{n-1}$ is that its $(n-1)$-cube facets naturally tile this hyperplane. We exploit this by recasting the unfolding in the language of lattices. Without loss of generality, we can situate a ridge unfolding of the n-cube so that the centroid of each facet occupies a point of the integer lattice $\mathbb{Z}^{n-1}$ of $\mathbb{R}^{n-1}$. To see the lattice structure manifest in the n-Roberts graph, we imbue the latter with a *coordinate system*: arbitrarily label the $2n-2$ edges of the n-Roberts graph incident to the base node with the directions

$$\{x_1, -x_1, x_2, -x_2, \ldots, x_{n-1}, -x_{n-1}\},$$

where edges incident to antipodal nodes get opposite directions.[1] Figure 5 shows examples of coordinate systems for the 3D, 4D, and 5D cases.

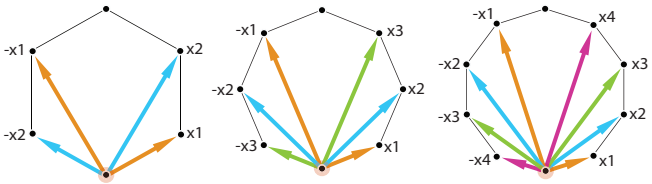


Figure 5: Coordinate systems for 3D, 4D, and 5D cubes.

These $n-1$ directions are mapped to the axes of the $\mathbb{R}^{n-1}$ hyperplane into which the n-cube unfolds. In particular, the $2n-2$ ridges of the n-cube incident to the base facet are in bijection with these coordinate directions, with opposite directions corresponding to parallel ridges of the facet. The roll keeps track of the combinatorics, whereas the coordinate system shows the direction of unfolding in the lattice. This is made precise:

Lemma 1 *Let T be a spanning tree of the n-Roberts graph with a coordinate system. The unfolding of the n-cube along T into $\mathbb{R}^{n-1}$ can be obtained by mapping T to the lattice $\mathbb{Z}^{n-1}$ through a sequence of rolls.*

Proof. Choose some base facet b of T and map it to some point $p_b \in \mathbb{Z}^{n-1}$. Let node c be adjacent to b along T with associated direction x from the coordinate system. The roll from b towards c maps node c to the point in $\mathbb{Z}^{n-1}$ that is adjacent to p_b in direction x. The four facet labels $\{b, c, b*, c*\}$ permute with the roll of the cube whereas the coordinate system directions are always anchored to the base facet. In particular, after the roll, facet $b*$ lies in the x direction with respect to the new base facet c, since the plane spanned by normal vectors n_b and n_c was rotated.

Given any node t of T, traverse the path between b and t through a series of rolls as described above; this maps all the nodes of T into $\mathbb{Z}^{n-1}$. To obtain the unfolding of the n-cube, simply replace each mapped point of the lattice with an $(n-1)$-cube. □

Example 2 *Figure 6 shows an unfolding of the 3-cube along a spanning path using Lemma 1. At each iteration, there is a roll of the Roberts graph and a direction of unfolding based on the given coordinate system. The unfolded facets are colored white, and the unfolded ridges*

[1] The isometry group of the cube acts simply transitively on these labelings. Thus, without loss of generality, we can choose a counterclockwise labeling of the edges in cyclic order.

become dashed-lines. Figure 7 showcases a 3-cube unfolding along a spanning tree. Given any two nodes of this tree, we unfold along the path between these nodes by rolls using Lemma 1. Figure 8 provides an example of an unfolding of the 4-cube along a spanning path.

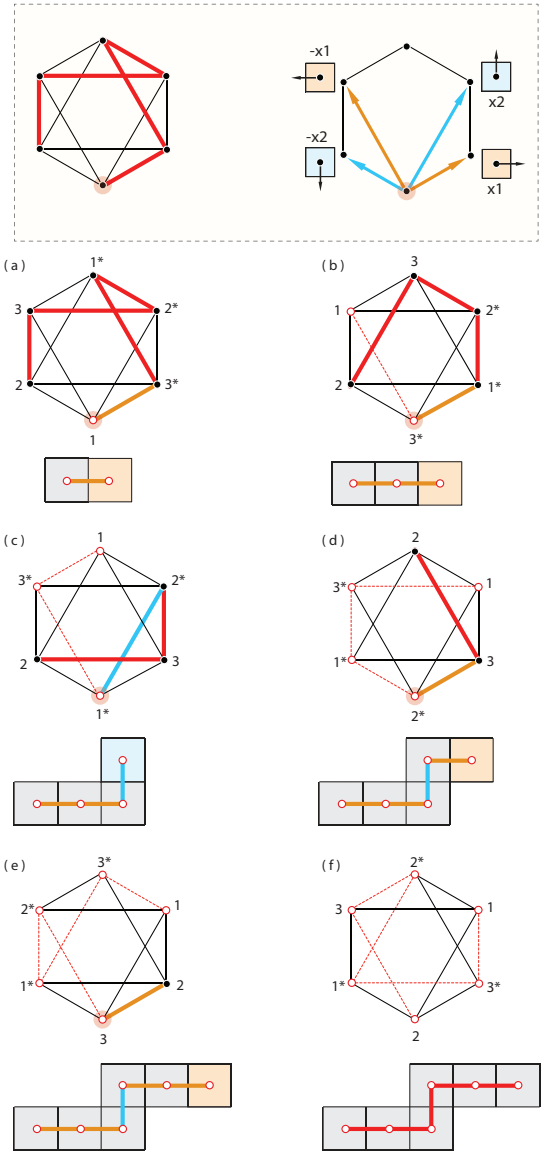


Figure 6: Unfolding a 3-cube along a spanning path.

Lemma 2 *Let T be a spanning tree of the n-Roberts graph with a coordinate system. If direction x is used in the unfolding along some path of T, direction $-x$ will not be used in the unfolding along this path.*

Proof. Assume we roll along a path in the x direction, moving the current base facet b into the $-x$ direction. Since b has now been visited, it cannot be used again in the unfolding. Thus, the only way a roll along direction $-x$ can occur is if b is rotated out of that direction. However, the only moves that can displace b are rolls along the x and $-x$ directions. The latter is not possible and the former simply replaces b with another visited facet, continuing to obstruct motion in the $-x$ direction. □

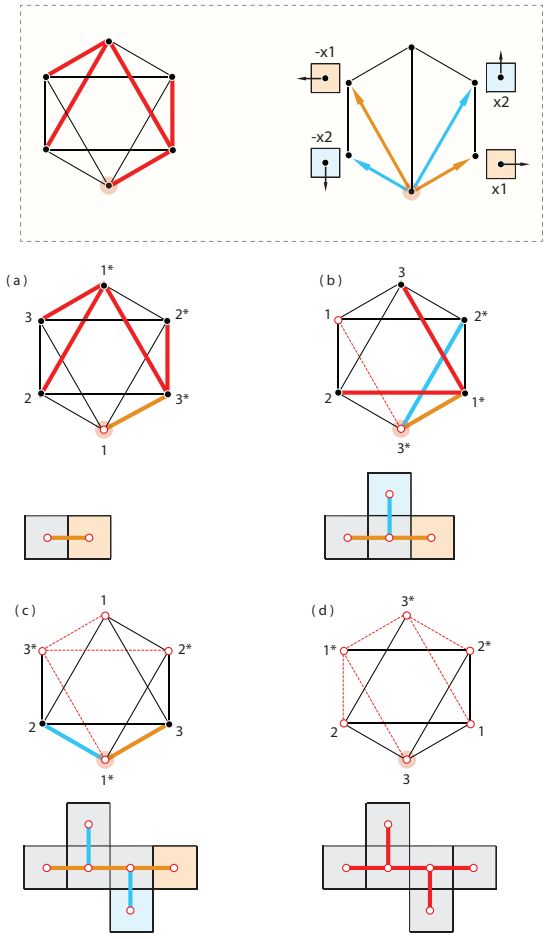


Figure 7: Unfolding a 3-cube along a spanning tree.

Example 3 *Figure 6(ab) shows an example where the first roll is in direction x_1, moving facet 1 into the $-x_1$ position, and facet 3^* into the base position. Since facet 1 has been visited, rolling in direction $-x_1$ is restricted. Another roll in Figure 6(bc) displaces 1 but simply replaces it with facet 3^*, which has now been visited.*

Theorem 3 *Every ridge unfolding of an n-cube yields a net.*

Proof. Consider an unfolding of the n-cube, given by a spanning tree T on the n-Roberts graph. By Lemma 2, antipodal directions will never appear in unfolding of paths. Thus, as the combinatorial distance between any two nodes of a path along the spanning tree T increases, the Euclidean distance of their respective facets in the hyperplane $\mathbb{R}^{n-1}$ (under the mapping to the integer lattice from Lemma 1) strictly increases. Since the facets

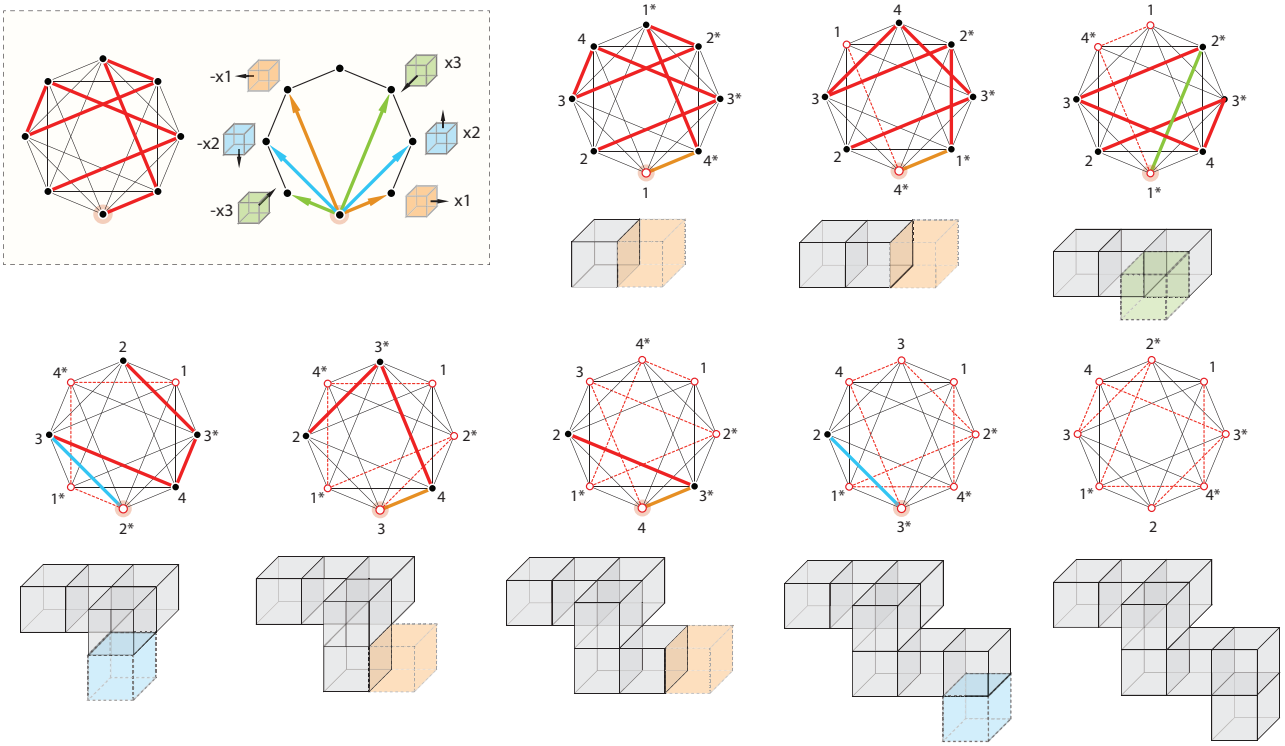


Figure 8: Unfolding a 4-cube along a spanning path.

in the unfolding along any path of T do not overlap, the unfolding of the entire tree T results in a net. □

3 Packings and Partitions

Having unfolded cubes into their nets, we now turn to packing these nets into boxes. A *box* (or *orthotope*) is the Cartesian product of intervals, and the *bounding box* of a net is the smallest box containing the net, with box sides parallel to the ridges of the net.

Definition 2 *An n-cube partition is an integer partition of $3n-2$ into $n-1$ parts, where each part is at least two.*

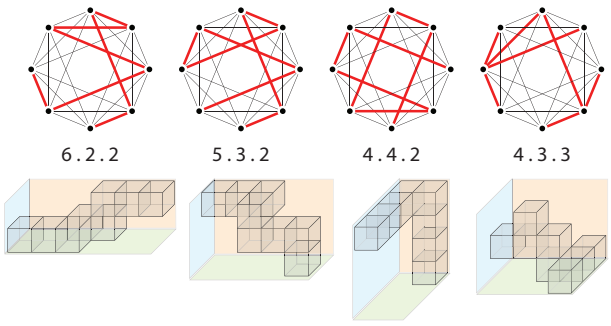


Figure 9: Spanning trees and bounding boxes.

Example 4 *Figure 9 displays four spanning trees of the 4-cube and their corresponding nets in bounding boxes. Notice that the dimensions of each bounding box form a 4-cube partition. In particular, these are all the possible 4-cube partitions. Theorem 4 below claims that all 261 nets of the 4-cube must fit into one of these four boxes.*

Theorem 4 *For every net of an n-cube, the dimensions of its bounding box is an n-cube partition.*

Proof. Each net of the n-cube has $2n$ facets that need to be unfolded in $\mathbb{R}^{n-1}$. Since each facet is an $(n-1)$-cube, the placement of the first facet in the unfolding contributes $n-1$ to the bounding box number of the net, one for each of its $n-1$ dimensions. We show that each of the remaining $2n-1$ facets of the unfolding increases the bounding box number by exactly 1, resulting in a total box number of $1 \cdot (n-1) + (2n-1) \cdot 1 = 3n-2$.

Suppose (by contradiction) that in the unfolding, the roll from facet b to adjacent facet c in direction x does not increase the bounding box number of the current net. Assume the ridge between b and c is supported by some hyperplane $\mathcal{H}$ of $\mathbb{R}^{n-1}$. Since the box number did not increase, there must be another facet (call it d) in the current unfolding that lies on the same side of hyperplane $\mathcal{H}$ as c. Thus, the unfolding of the path between facets c and d must have crossed $\mathcal{H}$ at least twice, moving along x in both the positive and negative directions, contradicting Lemma 2.

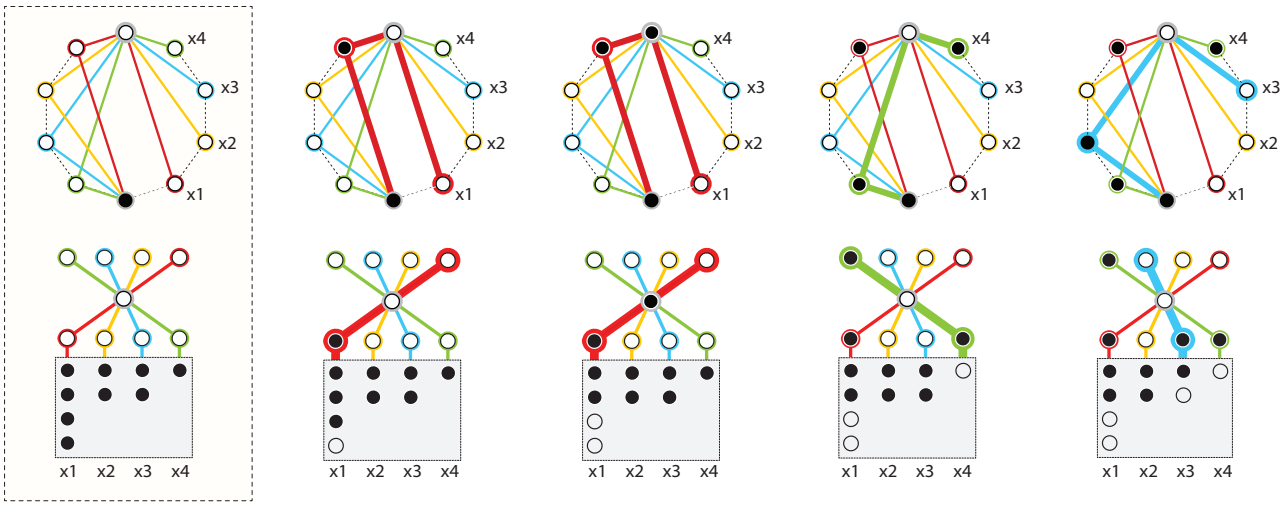


Figure 10: Rolls on the Roberts graphs reinterpreted as a token sliding game.

Finally, it needs to be shown that our cube will roll in all $n-1$ unfolding dimensions (satisfying the requirement that each part of a cube partition is a least two). But the cube net is a spanning tree of the Roberts graph, with the unfolding forced to visit all the nodes. And such visits can only be accomplished by rolling along each of the $n-1$ distinct directions. □

The converse of Theorem 4 also holds: given an integer partition of $3n-2$ into $n-1$ parts, there exists an unfolding of an n-cube whose bounding box dimensions match the partition. The remainder of this section is devoted to proving this result. As discussed earlier, the placement of the first facet in the unfolding of the n-cube contributes $n-1$ to the bounding box number. Thus, the cube partition can be reinterpreted as an integer partition of $2n-1$ (the remaining facets) into $n-1$ parts (the possible directions), with each part at least one. For such a partition, our task is to find a sequence of rolls along the $n-1$ directions so that the $2n-1$ facets are unfolded into their respective partitioned directions. Without loss of generality, we consider rolls only in the positive directions.

In order to construct cube unfoldings for such partitions, we reinterpret the Roberts graph as a token sliding game, with Figure 10 serving as a Rosetta stone. Consider the first column of this figure, where the n-Roberts graph on top is unraveled below into a game board with $n-1$ slides (appropriately color-coded). Here, the base node of the Roberts graph is replaced by our given partition, one for each direction, with the $2n-1$ positions represented by black tokens. The goal of this game is to move these tokens into the $2n-1$ empty slots on the game board above by a sequence of slides, corresponding to rolls of the Roberts graph.

The top row of Figure 10 shows a 5-cube rolling twice in the x_1 direction, followed by a roll in the x_4 direction, and a roll in the x_3 direction. The bottom row displays the corresponding tokens moving along their appropriate slides, leaving the partition box and occupying empty slots on the game board above. The features of the token game, inherited from the properties of rolls, are as follows:

1. Each roll of the Roberts graph in a particular direction slides *all* the tokens along that direction one place up.
2. When a token reaches the end of its slide (eg, direction x_4, as displayed by the fourth column of Figure 10), it can no longer use that direction.
3. The antipode to the base (topmost on the Roberts graph) acts as a *transfer point*, moving tokens from one directional slide into another.

Theorem 5 *For any n-cube partition, there exists a path unfolding of an n-cube whose bounding box dimensions matches the partition.*

Proof. We provide an unfolding algorithm by rolling along directions satisfying a given partition. Parts in the partition with more than one token are called *towers*, whereas parts with exactly one token are dubbed *singletons*. Begin by decomposing the $2n-1$ tokens into four groups:

1. The set S of tokens in the singletons.
2. The set B of bottom tokens in each tower.
3. The set T of top tokens in each tower.
4. The remaining set M of (middle) tokens.

It follows that $|T| = |B| = (n-1) - |S|$ and

$$|M| = (2n-1) - |T| - |B| - |S| = |S| + 1.$$

Example 5 *Figure 11 shows two distinct partitions of 15 tokens into 7 parts (when $n = 8$), labeled according to the terminology above. In these cases, it is clear that $|T| = |B|$ and $|M| = |S| + 1$.*

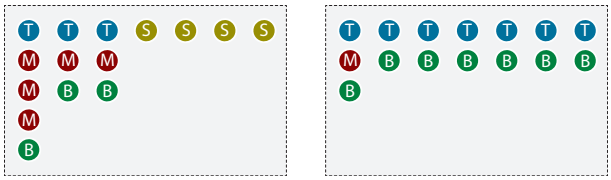


Figure 11: Two distinct partitions when $n = 8$.

Our algorithm is broken into three steps:

Step 1: *Perform one slide in each direction of a token from B.* This is possible since the transfer point is empty; see Figure 12(abc).

Step 2: *Perform alternating slides between tokens from M and S, starting and ending with M, until all such tokens depleted.* This is well-defined since $|M| = |S|+1$. Since the first position on the game board along any element of M already contains a token from Step 1, a slide along its direction moves this token into the transfer point; see Figure 12(d). Now, sliding a token of S fills the first and last positions along this directional track with tokens, making this direction unusable; see Figure 12(e). This is ideal, for S contains only one token in each direction. After alternating between M and S, depleting all elements of S, slide one final time along the last element of M, loading a token onto the transfer point; see Figure 12(f).

Step 3: *Perform one slide in each direction of a token from T.* Each slide moves the token of the transfer point to the end of the track, which replenishing the transfer point with another token. This fills all the positions, as these are the final elements in each tower; see Figure 12(ghi). □

Observation 1 *Theorem 5 shows that the n-cube can be unfolded into extremes: a long thin $2 \times \cdots \times 2 \times (n+2)$ box and a cubelike $3 \times \cdots \times 3 \times 4$ box, with a spectrum of sizes in between. It would be interesting to explore the* distribution *of cube partitions over all possible unfoldings of the n-cube.*

Observation 2 *Up to symmetry, there are 11 nets of the 3-cube and 261 nets of the 4-cube. For a general n-cube, it is an open problem to enumerate its distinct nets. The theorem above provides a (very weak) lower bound to this problem.*

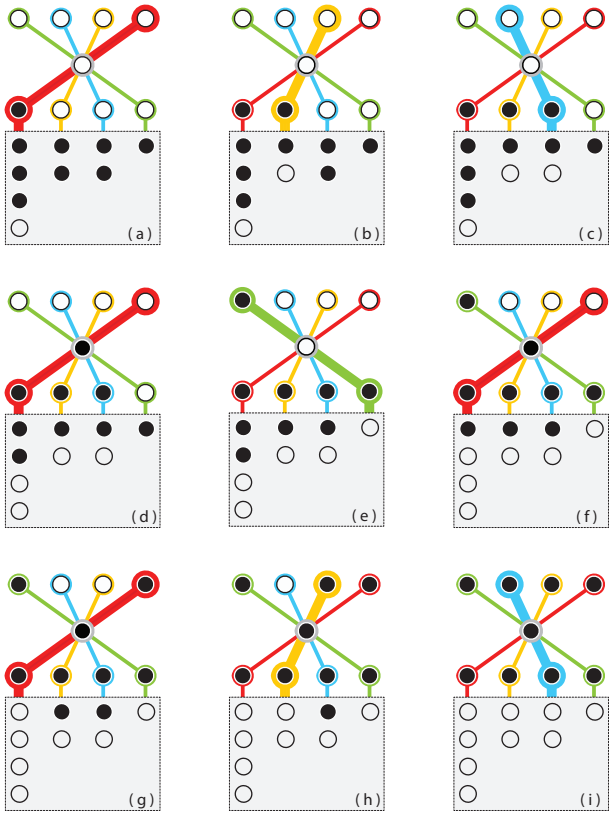


Figure 12: The partition slide algorithm.

4 Conclusion

The work of Horiyama and Shoji [3] show that every edge unfolding of the five Platonic solids results in a net. The higher-dimensional analogs of the Platonic solids are the regular convex polytopes: three classes of such polytopes exist for all dimensions (simplex, cube, orthoplex) and three additional ones only appear in 4D (24-cell, 120-cell, 600-cell). We have considered all unfoldings of cubes, and a similar result for simplices easily follows. We are encouraged to claim the following:

Conjecture 1 *Every ridge unfolding of a regular convex polytope yields a net.*

References

[1] A. D. Aleksandrov. *Intrinsic Geometry of Convex Surfaces*, Volume II (English translation), Chapman and Hall/CRC Press, 2005.

[2] F. Buekenhout and M. Parker. The number of nets of the regular convex polytopes in dimension ≤ 4, *Discrete Mathematics* **186** (1998) 69–94.

[3] T. Horiyama and W. Shoji. Edge unfoldings of Platonic solids never overlap, *Canadian Conference on Computational Geometry* (2011).

[4] E. Miller and I. Pak. Metric combinatorics of convex polyhedra: cut loci and nonoverlapping unfoldings, *Discrete and Computational Geometry* (2008) **39** 339–388.

[5] G. Shephard. Angle deficiencies of convex polytopes, *Journal of the London Mathematical Society* **43** (1969) 325–336.

[6] G. Shephard. Convex polytopes with convex nets, *Mathematical Proceedings of the Cambridge Philosophical Society* **78** (1975) 389–403.

[7] P. Turney. Unfolding the tesseract, *Journal of Recreational Mathematics* **17** (1984) 1–16.

Efficient Folding Algorithms for Regular Polyhedra

Tonan Kamata* Akira Kadoguchi† Takashi Horiyama‡ Ryuhei Uehara§

Abstract

We investigate the folding problem that asks if a polygon P can be folded to a polyhedron Q for given P and Q. Recently, an efficient algorithm for this problem is developed when Q is a box. We extend this idea to two different cases; (1) Q is a regular dodecahedron, and (2) Q is a convex polyhedron such that each face is formed by regular triangles. Combining the known result, we can conclude that the folding problem can be solved efficiently when Q is a regular polyhedron, also known as a Platonic solid.

1 Introduction

In 1525, the German painter Albrecht Dürer published his masterwork on geometry [7], whose title translates as "On Teaching Measurement with a Compass and Straightedge for lines, planes, and whole bodies." In the book, he presented each polyhedron by drawing a *net* for it: an unfolding of the surface to a planar layout by cutting along its edges. To this day, it remains an important open problem whether every convex polyhedron has a (non-overlapping) net by cutting along its edges. On the other hand, when we allow to cut anywhere, any convex polyhedron can be unfolded to a planar layout without overlapping. There are two known algorithms; one is called *source unfolding*, and the other is called *star unfolding* (see [6]).

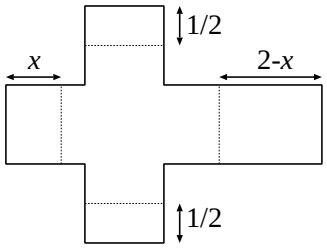


Figure 1: A Latin cross made by six unit squares. For any real number x with $0 < x < 1$, folding along dotted lines, we can obtain a doubly-covered fat cross.

*School of Information Science, JAIST, kamata@jaist.ac.jp
†School of Information Science, JAIST, s1810041@jaist.ac.jp
‡Faculty of Information Science and Technology, Hokkaido University, horiyama@ist.hokudai.ac.jp
§School of Information Science, JAIST, uehara@jaist.ac.jp

To understand unfolding, it is interesting to look at the inverse: what kind of polyhedra can be folded from a given polygonal sheet of paper? For example, the Latin cross, which is one of the eleven nets of a cube, can fold to 23 different convex polyhedra by 85 distinct ways of folding [6] and an infinite number of doubly covered concave polygons (Figure 1). Comprehensive surveys of folding and unfolding can be found in [6].

We investigate the folding problem when a polygon P and a polyhedron Q are given. That is, for a given polygon P and a polyhedron Q, the folding problem asks if P can fold to Q or not. This is a natural problem, however, there are few results so far. When Q is a regular tetrahedron, we have a mathematical characterization of its net [4]; according to this result, P should be a kind of tiling, and hence the folding problem can be solved efficiently. Abel et al. investigated the folding problem of bumpy pyramids [1]: For a given petal polygon P (convex n-gon B with n triangular petals), it asks if we can fold to a pyramid (with flat base B) or a convex bumpy pyramid by folding along a certain triangulation of B. In [1], they gave nontrivial linear time algorithms for the problem. Recently, the folding problem was investigated for the case that Q is a box. Some special cases were investigated in [8] and [11], and the problem for a box Q is solved in [10] in general case. The running time of the algorithm in [10] is $O(D^{11}n^2(D^5 + \log n))$, where D is the diameter of P. In these algorithms, Q is given as just a "box" without size, and the algorithms try all possible sizes. If Q is explicitly given, the running time of the algorithm in [10] is reduced to $O(D^7n^2(D^5 + \log n))$ time.

In this paper, we investigate two other cases. In the first case, we assume that Q is a regular dodecahedron. This is a very special case, however, it is one of the five regular polyhedra. The second case is that Q is a convex deltahedron whose faces consist of regular triangles. A deltahedron is said to be *strictly* convex if it is convex and no two adjacent faces are coplanar. It is known that there are eight strictly convex deltahedra[1]: a regular tetrahedron, a regular octahedron, a regular icosahedron, a triangular bipyramid, a pentagonal bipyramid, a snub disphenoid, a triangulated triangular prism, and a gyroelongated square bipyramid. In this paper, we also consider non-strictly-convex cases as a kind of deltahedron. That is, we allow each face to consist of coplanar

[1]See, e.g., https://en.wikipedia.org/wiki/Deltahedron.

regular triangles like a regular hexagon. Then there are infinite number of non-strictly-convex deltahedra. For these two cases, we give pseudo-polynomial time algorithms:

Theorem 1 *Let P be a simple polygon with n vertices. We denote by L the perimeter of P. Then the folding problem of a regular dodecahedron from P can be solved in $O(n^2(n+L)^4)$ time.*

Theorem 2 *Let P be a simple polygon with n vertices of perimeter L. Let Q be a non-concave deltahedron[2] with m vertices. Then the folding problem of Q from P can be solved in $O(n^2m(L+n)^2)$ time.*

Combining with the result in [10], we have the following:

Corollary 3 *The folding problem for the five regular polyhedra (also known as Platonic solids) can be solved in pseudo-polynomial time.*

We here note that we use real RAM model, and the time complexity is evaluated by the number of mathematical operations.

2 Preliminaries

We first state the folding problem: the input is a polygon $P = (p_0, p_1, \ldots, p_{n-1}, p_n = p_0)$ and a polyhedron Q, and the problem asks if P can fold to Q or not. Let x_i and y_i be the x-coordinate and y-coordinate of a point p_i, respectively. We assume the real RAM model for computation; each coordinate is an exact real number, and the running time is measured by the number of mathematical operations.

When Q is a regular dodecahedron, we do not need to give it explicitly as a part of input. The length of the edges of Q can be computed from the area of P. Without loss of generality, we assume that the length of an edge of Q is 1. When Q is a non-concave deltahedron, Q is represented in the standard form in computational geometry (see [5]). Precisely, Q consists of vertices $q_i = (x_i, y_i, z_i)$, edges $\{q_i, q_j\}$, and faces $f_1, \ldots, f_k$, where each f_i is represented by a cycle of vertices in counterclockwise-order in relation to the normal vector of the face.

Let Q be a convex polyhedron. Let q be a vertex of Q. The *curvature* at q is the angle defined by the value $360^\circ - a$, where a is the angle surrounding q when it is unfolded on a plane.

Theorem 4 ([Gauss-Bonnet Theorem]) *The total sum of the curvature of all vertices of a convex polyhedron is* 720°.

[2]For simplicity, we call "non-concave deltahedron" a polyhedron that is either a convex deltahedron or a non-strictly-convex deltahedron.

See [6, Sec. 21.3] for details.

Let Q be a convex polyhedron. A *development* of Q results when we cut Q along a set of polygonal lines, unfold on a plane, and obtain a polygon P. We assume that any cut ends at a point with curvature less than 360°. Otherwise, since Q is convex, it makes a redundant cut on P, which can be eliminated (the proof can be found in [9, Theorem 3]). The polygon P is called a *net* of Q if and only if P is a connected simple polygon, i.e., without self-overlap or hole. Let T be the set of cut lines on Q to obtain a net P. Then the following is well known (see [6, Sec. 22.1.3] for details):

Theorem 5 *T is a spanning tree of the vertices of Q.*

2.1 Properties of Unfolding

A *tetramonohedron* is a tetrahedron that consists of four congruent triangles. This polyhedron is exceptional in the context of unfolding. To avoid this case, we first show the following lemma.

Lemma 6 *Let Q be a convex polyhedron. Then Q is a tetramonohedron if and only if the curvature of every vertex is* 180°.

Proof. If Q is a tetramonohedron, by its symmetric property, each vertex q consists of three distinct angles of a congruent triangle. Thus the curvature at q is 180°. In order to show the opposite, we assume that every vertex of a polyhedron Q has curvature 180°. Then, by Theorem 4, Q has four vertices. Let q_0, q_1, q_2, q_3 be these four vertices. We cut along three straight lines q_0q_1, q_0q_2, q_0q_3 on Q, respectively. Since the curvature at any point on Q except q_0, q_1, q_2, q_3 is 360°, we can take three non-crossing straight lines from q_0 to q_1, q_2, and q_3 on Q and they are the shortest lines from q_0 to them. By developing Q from q_0 along these three cut lines, we obtain a polygon $P = (q_0, q_1, q_0', q_2, q_0'', q_3, q_0)$. Then, by assumption, curvatures at q_1, q_2, q_3 are all 180°. That is, P is a triangle with three vertices q_0, q_0', and q_0''. Moreover, each edge of the triangle consists of two cut lines which form an edge on Q. Therefore, q_1, q_2, and q_3 are all the middle points of three edges q_0q_0', $q_0'q_0''$, and $q_0''q_0$ of the triangle P, respectively. Thus all four triangles $q_0q_1q_3$, $q_1q_0'q_2$, $q_3q_2q_0''$, and $q_2q_3q_1$ are congruent, which implies that Q is a tetramonohedron. □

We note that there is a mathematical characterization of a net of a tetramonohedron by a tiling (see [3, Chapter 3][3]). Using this property, the folding problem for a tetramonohedron Q can be solved in pseudo-polynomial time (the details are omitted in this conference version).

In this paper, the following theorem is useful:

[3]In [3], a tetramonohedron is called an *isotetrahedron*.

Theorem 7 *Let Q be a convex polyhedron that is not a tetramonohedron, and P be a net of Q. Then (1) all vertices of Q appear on the boundary of P, and (2) P has at least two vertices of angle not equal to* 180° *that correspond to distinct vertices of Q.*

Proof. A vertex of Q has positive curvature. Hence it cannot correspond to an interior point of P. Thus we have the claim (1). Now we focus on the claim (2). We first show that Q has at least two vertices of curvature not equal to 180°. Let $q_0, \ldots, q_k$ be the vertices of Q. Since Q is a convex polyhedron, $k \geq 3$. When $k = 3$, the only possible solid is a doubly covered triangle. Then it is easy to see that Q satisfies the claim (2). If $k > 4$, by Theorem 4, it is easy to see that at least two vertices have curvature not equal to 180°. Thus we consider the case that $k = 4$. By Lemma 6, since Q is not a tetramonohedron, four vertices cannot have curvatures equal to 180°. By Theorem 4, it is impossible that three vertices have curvature 180°, and one vertex does not. Thus Q has at least two vertices q and q' of curvature not equal to 180°. Then, by (1), q and q' correspond to distinct vertices of P. Now consider the set S_q of vertices of P that are glued together to form q. Then, since the curvature of q is not equal to 180°, at least one of the elements in S_q has an angle not equal to 180°. Thus q produces at least one vertex on P of angle not equal to 180°. We have another vertex on P produced by q' by the same argument. Thus we have the theorem. □

3 Algorithms

In this section, we first describe the common outline of algorithms and show the details for each case.

3.1 Outline of Algorithm

The outline of our algorithm is simple:

That is, the algorithm checks all possible combinations of pairs $\{p_i, p_{i'}\}$ and q_j. By Theorem 7, if Q can be folded from P, there are at least two vertices $p_i, p_{i'}$ of P that correspond to $q_j, q_{j'}$ of Q for some $q_{j'}$, respectively, with $i \neq i'$ and $j \neq j'$. Hereafter, we assume that the vertex p_0 of P corresponds to the vertex q_0 of Q, and p_i of P corresponds to the vertex q_j of Q, respectively, without loss of generality.

The key point is how to decide the relative orientation of Q and P, which has an influence of time complexity of the algorithm. Intuitively, for this issue, we also try all possible cases. The time complexity (or the number of trials) is different depending on the shape of Q. For the remainder of Section 3.1, we assume that the orientation of Q relative to P is fixed. We give the details of the two phases in the algorithm.

Input : A polygon $P = (p_0, p_1, \ldots, p_{n-1}, p_0)$ and a convex polyhedron Q
Output: All ways of folding P to Q (if one exists)
Let $\{q_0, \ldots, q_{m-1}\}$ be the set of vertices of Q;
foreach *pair of two vertices* $\{p_i, p_{i'}\}$ *of* P **do**
  foreach *vertex* q_j *of* Q **do**
    Check if Q is reachable from p_i to $p_{i'}$ on P by stamping Q so that p_i and $p_{i'}$ correspond to q_j and $q_{j'}$, respectively, for some $q_{j'}$ with $j \neq j'$ on Q;
    Check if P is a net of Q by folding and gluing P based on the partition of Q by stamping;
  end
end

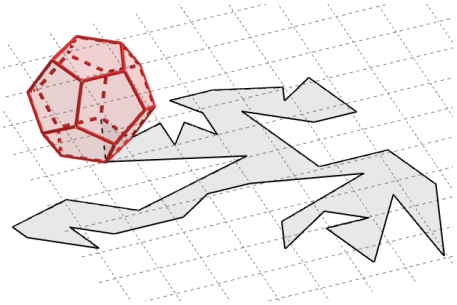


Figure 2: An example of stamping.

3.1.1 The first phase: stamping

When we have the correspondence of p_0 and q_0, we imagine placing Q on P at $p_0 = q_0$. Then we "roll" Q on P from the initial position in the depth first search (DFS) manner. This idea is called *stamping* in [2]. In [2], Akiyama rolled a regular tetrahedron on a plane as a *stamper* and obtained a tiling by the stamping. The key property of the stamping in [2] is that a regular tetrahedron has the same orientation and position when it returns to the original position, no matter what the route was. Therefore, the cut lines of any net on the surface of a regular tetrahedron tile the plane as a p2 tiling (see [2] for the details of p2 tiling).

In our case, we use a polyhedron Q as a stamper on P. We first put Q on P so that $p_0 = q_0$. Then the intersection of P and F_0 can be a set of polygons. Among them, we take a polygon P_0 that contains $p_0 = q_0$ as a vertex of P_0. (If two or more such polygons exist, we can take any one of them.) Then, if P_0 properly contain a vertex of F_0, we reject this position. It is not difficult to see that F_0 should have a part of edges e of F_0 that is inside of P. Then we can roll Q along the edge e, and the next face F_1 of Q is put on P, which shares the edge e with F_0. Then the intersection of F_1 and P

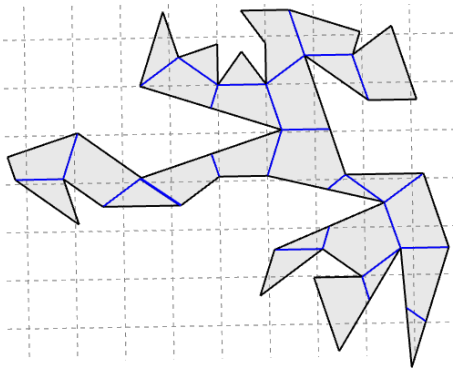


Figure 3: Tree structure of P: Each face of Q is cut into "patches". Then the adjacent relationship of the patches induces a tree on P.

gives the new set of polygons. Among them, we take P_1 that shares a part of e with P_0. (If two or more such polygons exist, we can take any one of them again.) If P_1 properly contains a vertex of F_1, this stamping fails. Repeating this process, we can stamp Q on P. See Figure 2 for a simple example. Precise description of this stamping can be found in [10]. In our algorithm, $p_{i'}$ can be used as a check point. if $p_{i'}$ does not match a certain $q_{j'}$, this stamping fails immediately.

Then, in our context, if P is a net of Q, we have the following properties by Theorem 7(1): (1) all (copies of) vertices of Q are on the boundary of P, and (2) for each vertex q_j of Q, the curvatures corresponding to the points on the boundary of P sum up to the angle at q_j on Q. In other words, no vertex of Q exists inside of P. On the other hand, since P has no hole, each point in P is stamped by a face of Q (or an edge of Q) exactly once. We will use the tree structure of P defined as follows (Figure 3). Each face of Q is cut into "patches" when it is developed to P. In other words, P is partitioned into patches by the edges of Q (or folding lines of P). On the tree, the patches of P correspond to the vertices, and two vertices are adjacent if and only if the corresponding patches share a part of an edge of positive length on Q. Then since P is a simple polygon (without hole)[4] and all vertices of Q are on the boundary of P, the resulting graph induces a tree. Essentially, the sequence of stamping of Q on P is a search algorithm on this tree structure, or it gives the partition of P into the patches by stamping of Q. We can traverse the tree by rolling Q. Hereafter, for simplicity, we assume that the algorithm traverses this tree by rolling Q on P in a DFS manner. Formally, we have the following lemma:

Lemma 8 ([10]) *Assume that Q gives us a feasible stamping of P. We also assume that P is a net of Q. Then, the stamping gives us a mapping from each point p in P to a point q in a face of Q, or p is glued to the point q in a face of Q (except on the edges of Q; in this case, the point p of P is glued to a point q on the edge of Q). That is, the stamping gives us a partition of P by the edges of Q.*

In [10], they give a proof of Lemma 8 when Q is a box. However, the arguments in [10] use only the fact that Q is convex. Thus, we can obtain Lemma 8 as a natural extension of the result in [10].

We note that each feasible stamping gives us all the vertices q_i of Q on the boundary of P. Therefore, we can check if each vertex q_i has a certain curvature in total in linear time of n in this phase. Therefore, after the first phase, we know that P is partitioned into patches of faces of Q with their correspondence (i.e., each patch knows its corresponding face of Q) and each vertex q_i has a correct curvature in total.

3.1.2 The second phase: gluing

After the first phase, we obtain a set of patches of faces of Q, and corresponding vertices produce certain curvatures, however, we have not yet checked if each set of patches really forms the corresponding face of Q by gluing. In other word, we have to check if the mapping in 8 is one-to-one mapping or not. Thus, in the second phase, we check if we fold to Q by P along the crease lines computed in the first phase. Hereafter, we sometimes consider the polygon $P = (p_0, p_1, \ldots, p_{n-1}, p_0)$ consists of vectors $\overrightarrow{p_0p_1}, \overrightarrow{p_1p_2}, \ldots, \overrightarrow{p_{n-1}p_0}$ for the sake of simplicity. Then we can deal with "gluing of two edges" by an operation of vectors. For example, we assume that we glue two edges p_0p_1 and p_0p_{n-1}. Then, we have three cases after gluing:
(1) $|p_0p_1| > |p_0p_{n-1}|$: We obtain $\overrightarrow{p_{n-1}p_1}$ such that $|p_{n-1}p_1| = |p_0p_1| - |p_0p_{n-1}|$.
(2) $|p_0p_1| < |p_0p_{n-1}|$: We obtain $\overrightarrow{p_{n-1}p_1}$ such that $|p_{n-1}p_1| = |p_0p_{n-1}| - |p_0p_1|$.
(3) $|p_0p_1| = |p_0p_{n-1}|$: We obtain $p_{n-1} = p_1$.

Recall that if P is a net of Q, the set of line segments of cut on Q forms a spanning tree T (Theorem 5). Moreover, each edge of T appears twice on the boundary of P. Now we know that $v_0 = p_0$ is a vertex of Q, and the orientation of Q with respect to P is fixed, then along the tree produced by stamping, we can check the gluing one by one from the leaves of T. Since T is a tree, we always have a pair of edges to be glued. The details of this part can be found in [8].

3.2 Case 1: Q is a Regular Dodecahedron

In this section, we assume that Q is a regular dodecahedron and the length of each edge is 1. Since the area

[4] Precisely speaking, it is enough that P is a *weakly* simple polygon (see `https://en.wikipedia.org/wiki/Simple_polygon#/media/File:Weakly_simple_polygon.svg`) to obtain this tree structure.

of a pentagon of unit edge is $\frac{\sqrt{25+10\sqrt{5}}}{4}$, we assume the area of P is $12\times\frac{\sqrt{25+10\sqrt{5}}}{4} = 3\sqrt{25+10\sqrt{5}}$ without loss of generality. Since we know Q, the input of this problem is just a polygon $P = (p_0, p_1, \ldots, p_{n-1}, p_0)$ of area $3\sqrt{25+10\sqrt{5}}$, and we will decide if P can be folded to a unit-size regular dodecahedron Q. By the argument in Section 3.1, we also know that two vertices p_0 and p_i of P correspond to two distinct vertices, say q_0 and q_j, of Q.

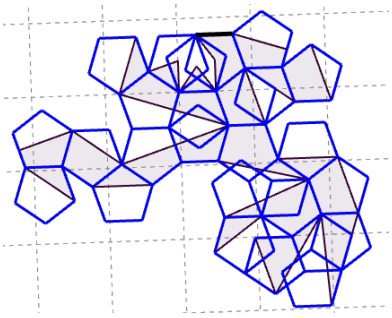


Figure 4: An example of overlapping stamping. Some pentagons are overlapping by stamping of Q along a feasible net P.

Stamping By assumption, Q can reach from p_0 to p_i on P by stamping Q such that p_0 and p_i are corresponding to two different vertices of Q. By rotation of P, we have a sequence of regular pentagons $(\hat{P}_0, \hat{P}_1, \ldots, \hat{P}_k)$ such that (1) $\hat{P}_0$ contains the edge joining points $p_0 = (0,0)$ and $(1,0)$ as its base edge, (2) $p_i = (x_i, y_i)$ is a vertex of $\hat{P}_k$, and (3) two consecutive pentagons $\hat{P}_{j'}$ and $\hat{P}_{j'+1}$ share an edge for each j' with $0 \le j' < k$. We note that two consecutive pentagons do not overlap (without their shared edge), but nonconsecutive pentagons can overlap by stamping (see Figure 4).

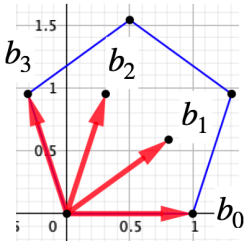


Figure 5: Four unit vectors for a unit pentagon.

Intuitively, if we put Q on P with a proper relative angle, Q can be unfolded along P, and we can reach from p_0 to p_i by traversing the edges of these regular pentagons. If we consider each edge of the pentagons as a unit vector, this traverse can be represented by a linear combination of the following four vectors: $\overrightarrow{b_0} = (0,1)$, $\overrightarrow{b_1} = (\cos\frac{\pi}{5}, \sin\frac{\pi}{5})$, $\overrightarrow{b_2} = (\cos\frac{2\pi}{5}, \sin\frac{2\pi}{5})$, and $\overrightarrow{b_3} = (\cos\frac{3\pi}{5}, \sin\frac{3\pi}{5})$. Note that $(\cos\frac{4\pi}{5}, \sin\frac{4\pi}{5}) = -\overrightarrow{b_0} + \overrightarrow{b_1} - \overrightarrow{b_2} + \overrightarrow{b_3}$. Thus, Q can be folded from P only if we have four integers B_0, B_1, B_2, B_3 such that

$$\overrightarrow{(p_0, p_i)} = B_0\overrightarrow{b_0} + B_1\overrightarrow{b_1} + B_2\overrightarrow{b_2} + B_3\overrightarrow{b_3},$$

and hence

$$\left|\overrightarrow{(p_0, p_i)}\right| = \left|B_0\overrightarrow{b_0} + B_1\overrightarrow{b_1} + B_2\overrightarrow{b_2} + B_3\overrightarrow{b_3}\right|.$$

It is clear that $|B_0| + |B_1| + |B_2| + |B_3|$ is at most L, where $L = \sum_{i=0}^{n-1}\left|\overrightarrow{(p_i, p_{i+1})}\right|$ is the perimeter of P. Thus, in our algorithm, we check $O((L+n)^4)$ combinations of four integers B_0, B_1, B_2, B_3. For each possible integers B_0, B_1, B_2, B_3, we can compute $p_i = (x_i, y_i)$ by rotation of P. After putting P on the proper place so that $p_0 = (0,0)$ and $p_i = (x_i, y_i)$, we perform the stamping of Q on P and obtain the partition of P. We here note that we use the commutative law of vectors. Thus the first relative position of Q is one of four positions along $\overrightarrow{b_0}, \overrightarrow{b_1}, \overrightarrow{b_2}, \overrightarrow{b_3}$. For each position, we perform the second phase for checking gluing.

Gluing By stamping of Q on P, P is partitioned into patches $\mathcal{P} = \{P_0, P_1, \ldots, P_{h-1}\}$. More precisely, P_0 is the intersection of P and Q on an initial position such that $p_0 = q_0 = (0,0)$. Since it is a valid stamping, there are no vertices of Q inside of P_0. When we roll Q on P along an edge e of Q, the other face is on P, and we have the next patch P_1 by the intersection of P and Q. We note that P_1 is the component of the intersection of P and Q that contains the edge e (can be partial, but not just a point). That is, even if we have non-empty intersection polygons of P and Q, we do not count them if they do not include any non-empty set of e. By repeating this process in the DFS manner, we obtain a set of patches $\mathcal{P} = \{P_0, P_1, \ldots, P_{h-1}\}$ that forms a partition of P. We now define the graph $T = (\mathcal{P}, E)$ by $E = \{\{P_i, P_j\} \mid P_i, P_j$ share a non-empty edge inside $P\}$. Intuitively, P_i and P_j share a non-empty edge inside P, and P_j is put in $\mathcal{P}$ by rolling Q from P_i along the edge or vice versa. As discussed in Section 3.1.1, the graph T is a tree. For notational convenience, we consider P_0 is the root of T, and the elements in $\mathcal{P}$ are numbered from P_0 in the way of the DFS manner.

First, we glue P_0 on Q so that the corresponding vertex p_0 on P (or P_0) comes to a vertex q_0 of Q. Then the gluing process is done on Q from P_0 in the DFS manner. This can be done in $O(\sum_{i=0}^{h-1}|P_i|)$ time, where $|P_i|$ is the number of vertices of P_i. The number $(h-1)$ is given by the number of stampings made by Q. For this number, we have the following upper bound:

Theorem 9 $h = O(L+n)$.

Proof. The number of stampings of Q on P is given by the total number of visits of each P_i. On the other hand, h is the number of P_is. Thus, precisely, $(h-1)$ is the number of the first visiting each P_i by Q except P_0. The stamping of Q is done along the DFS tree. Therefore, since each edge of the DFS tree is traversed twice, the number of stampings made by Q is $2(h-1)$. Thus h is proportional to the number of stampings.

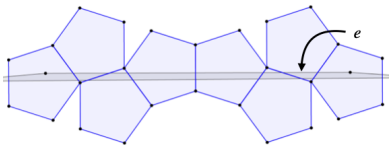


Figure 6: An edge e can be covered by $O(|e|)$ pentagons since each angle of a pentagon is 108°.

Let e be an edge of P. By stamping of Q along the edge e, since each pentagonal face of Q has unit size, the number of pentagons $\hat{P}_i$ to cover e is $O(|e|)$, where $|e|$ is the length of e (Figure 6; see also [10]). Thus, we have the number of pentagonal faces of Q as stamps to cover all the edges e of P is $O(L+n)$ in total. Therefore, we obtain $h = O(L+n)$. □

Time complexity Now we consider the time complexity of our algorithm for a regular dodecahedron. For a given polygon $P = (p_0, p_1, \ldots, p_{n-1}, p_0)$, the algorithm first guesses all possible combinations of $(p_i, p_{i'})$, which produce $O(n^2)$ cases. We here note that we essentially have one way of choosing q_0 by symmetry of Q. For this q_0, we have a constant number (precisely, it is 7) of cases of q_j. Thus we do not need to consider this constant factor for a regular dodecahedron. For each pair $(p_i, p_{i'})$, we construct a vector $\overrightarrow{(p_i, p_{i'})} = B_0\overrightarrow{b_0} + B_1\overrightarrow{b_1} + B_2\overrightarrow{b_2} + B_3\overrightarrow{b_3}$ by finding B_0, B_1, B_2, B_3. This step checks $O((L+n)^4)$ combinations if we check all combinations. However, when B_0, B_1, B_2 are fixed, since $\left|\overrightarrow{(p_0, p_i)}\right| = \left|B_0\overrightarrow{b_0} + B_1\overrightarrow{b_1} + B_2\overrightarrow{b_2} + B_3\overrightarrow{b_3}\right|$, we have two possible values depending on $B_3 \geq 0$ or $B_3 < 0$, and they can be computed in a constant time. Thus it is enough to check $O((L+n)^3)$ combinations. For each case, the algorithm performs stamping of Q. During the stamping, we check if each vertex of a face of Q is inside P or not. It is done along the traverse of the tree in DFS manner, and hence it can be done in $O(n)$ time in total. Thus the running time of stamping is $O(n+h)$, where $(h-1)$ is the number of stampings. By Theorem 9, we have $h = O(L+n)$. After the (valid) stamping, we obtain a partition $\mathcal{P} = \{P_0, P_1, \ldots, P_{h-1}\}$ of P. Checking the gluing of elements in $\mathcal{P}$ onto Q takes $O(\sum_{i=0}^{h-1} |P_i|)$ time. Since the tree $T = (\mathcal{P}, E)$ has h vertices and $h-1$ edges, we have $\sum_{i=0}^{h-1} |P_i| = O(n+h)$. Therefore, in total, the algorithm runs in $O(n^2(n+L)^4)$ time, which completes the proof of Theorem 1.

3.3 Case 2: Q is a Non-concave Deltahedron

In this section, we assume that Q is a non-concave deltahedron. We assume that each face of Q consists of some unit regular triangles; each unit triangle has three edges of unit length 1 and area $\frac{\sqrt{3}}{4}$. Let t be the total number of unit triangles on the surface of Q. That is, the surface area of Q is $\frac{\sqrt{3}}{4}t$. Let $\{q_0, q_1, \ldots, q_{m-1}\}$ be the set of vertices of Q. We assume that (1) the set of faces $\{F_0, F_1, \ldots, F_{f-1}\}$ of Q is given, where f is the number of faces of Q, (2) each vertex q_j has its coordinate (x_j, y_j, z_j), and (3) each face has its vertices in clockwise order. The basic idea of the algorithm is the same as in Section 3.2; we here consider the differences.

We still have the property that we can reach from p_0 to p_i on P by stamping Q on it. However, now we have $O(n^2 m)$ combinations for pairs of pair $(p_i, p_{i'})$ and q_j. Hereafter, we assume that the vertex p_i of P forms a vertex q_j on Q and the vertex $p_{i'}$ forms some vertex $q_{j'}$ on Q. In the same argument in the pentagonal case, for two vectors $\overrightarrow{b_0} = (1, 0)$ and $\overrightarrow{b_1'} = (\cos\frac{\pi}{3}, \sin\frac{\pi}{3}) = (\frac{1}{2}, \frac{\sqrt{3}}{2})$, Q can be folded from P only if we have two integers B_0' and B_1' such that

$$\left|\overrightarrow{(p_i, p_{i'})}\right| = \left|B_0'\overrightarrow{b_0} + B_1'\overrightarrow{b_1'}\right|.$$

We again have that $|B_0'| + |B_1'|$ is at most L, and hence we have $O((L+n)^2)$ combinations to be checked. However, once we fix B_0', then B_1' has two possible values. Thus this step requires $O(L+n)$ combinations. Each partition of P by stamping of Q takes $O(L+n)$ time by the same argument in the case of dodecahedron.

For gluing, almost all arguments are the same as pentagonal case since they do not use the fact that the shape of a face is a pentagon. The only difference is that we stamp all (possibly different) faces of Q; this fact gives us an additional lower bound f of the number of stampings. Therefore, the time complexity of our second algorithm for a non-concave deltahedron is $O(n^2 m(n+L)(n+f+L))$. Here, by the Euler characteristic, we have $f = 2+e-m$, where e is the number of edges of Q. When we consider Q as a graph, it is a planar graph, which implies that $e = O(m)$, or $f = O(m)$. By Theorem 4, Q has at most four vertices of curvature 180°. Thus we have $m = O(n)$. Therefore, the time complexity of the second algorithm is $O(n^2 m(n+L)^2)$, which completes the proof of Theorem 2.

4 Concluding Remarks

In this paper, we developed efficient folding algorithms for some classes of convex polyhedra. The next step is the extension to general convex polyhedra. In this case,

the number of unit vectors cannot be bounded by a constant number. Thus we need some different approaches. Additional future work is the extension to non-convex polyhedra. In this case, we cannot use Theorem 4 or Theorem 7 any longer. Is there any reasonable class of non-convex polyhedra such that the folding problem can be solved efficiently?

References

[1] Z. R. Abel, E. D. Demaine, M. L. Demaine, H. Ito, J. Snoeyink, and R. Uehara. Bumpy Pyramid Folding. *Computational Geometry: Theory and Applications*, 75:22–31, 2018.

[2] J. Akiyama. Tile-Makers and Semi-Tile-Makers. *The Mathematical Association of Amerika*, Monthly 114:602–609, August-September 2007.

[3] J. Akiyama and K. Matsunaga. *Treks into Intuitive Geometry*. Springer, 2015.

[4] J. Akiyama and C. Nara. Developments of Polyhedra Using Oblique Coordinates. *J. Indonesia. Math. Soc.* , 13(1):99–114, 2007.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 1998.

[6] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007.

[7] A. Dürer. *Underweysung der messung, mit den zirckel un richtscheyt, in Linien ebnen unnd gantzen corporen*. 1525.

[8] T. Horiyama and K. Mizunashi. Folding Orthogonal Polygons into Rectangular Boxes. In *In Proc. of 19th Japan-Korea Joint Workshop on Algorithms and Computation (WAAC 2016)*, 2016.

[9] J. Mitani and R. Uehara. Polygons folding to plural incongruent orthogonal boxes. In *20th Canadian Conference on Computational Geometry (CCCG 2008)*, pages 31–34, 2008.

[10] K. Mizunashi, T. Horiyama, and R. Uehara. Efficient Algorithm for Box Folding. *Journal of Graph Algorithms and Applications*, 24(2):89–103, February 2020.

[11] D. Xu, T. Horiyama, T. Shirakawa, and R. Uehara. Common Developments of Three Incongruent Boxes of Area 30. *Computational Geometry: Theory and Applications*, 64:1–17, 2017. DOI:10.1016/j.comgeo.2017.03.001.

Vertex-Transplants on a Convex Polyhedron

Joseph O'Rourke

Abstract

Given any convex polyhedron $\mathcal{P}$ of sufficiently many vertices n, and with no vertex's curvature greater than π, it is possible to cut out a vertex, and paste the excised portion elsewhere along a vertex-to-vertex geodesic, creating a new convex polyhedron $\mathcal{P}'$ of $n+2$ vertices.

1 Introduction

The goal of this paper is to prove the following theorem:

Theorem 1 *For any convex polyhedron $\mathcal{P}$ of $n > N$ vertices, none of which have curvature greater than π, there is a vertex v_0 that can be cut out along a digon of geodesics, and the excised surface glued to a geodesic on $\mathcal{P}$ connecting two vertices v_1, v_2. The result is a new convex polyhedron $\mathcal{P}'$ with $n+2$ vertices. $N = 16$ suffices.*

I conjecture that N can be reduced to 4 so that the theorem holds for all convex polyhedra with the stated curvature condition. Whether this curvature condition is necessary is unclear.

I have no particular application of this result, but it does raise interesting questions (Sec. 8), including: What are the limiting shapes if *vertex-transplants* are repeated indefinitely?

2 Examples

Before detailing the proof, we provide several examples.

We rely on Alexandrov's celebrated gluing theorem [Ale05, p.100]: If one glues polygons together along their boundaries[1] to form a closed surface homeomorphic to a sphere, such that no point has more than 2π incident surface angle, then the result is a convex polyhedron, uniquely determined up to rigid motions. Although we use this theorem to guarantee that transplanting a vertex on $\mathcal{P}$ creates a new convex polyhedron $\mathcal{P}'$, there is as yet no effective procedure to actually construct $\mathcal{P}'$, except when $\mathcal{P}'$ has only a few vertices or special symmetries.

In the examples below, we use some notation that will not be fully explained until Sec. 3.

[1] To "glue" means to identify boundary points.

Regular Tetrahedron. Let the four vertices of a regular tetrahedron of unit edge length be v_1, v_2, v_3 forming the base, and apex v_0. Place a point x on the v_3v_0 edge, close to v_3. Then one can form a digon starting from x and surrounding v_0 with geodesics γ_1 and γ_2 to a point y on $\triangle v_1v_2v_0$, with $|\gamma_1| = |\gamma_2| = 1$. See Fig. 1(a,b). This digon can then be cut out and its hole sutured closed. The removed digon surface can be folded to a doubly covered triangle, and pasted into edge v_1v_2. The resulting convex polyhedron guaranteed by Alexandrov's Theorem is a 6-vertex irregular octahedron $\mathcal{P}'$.

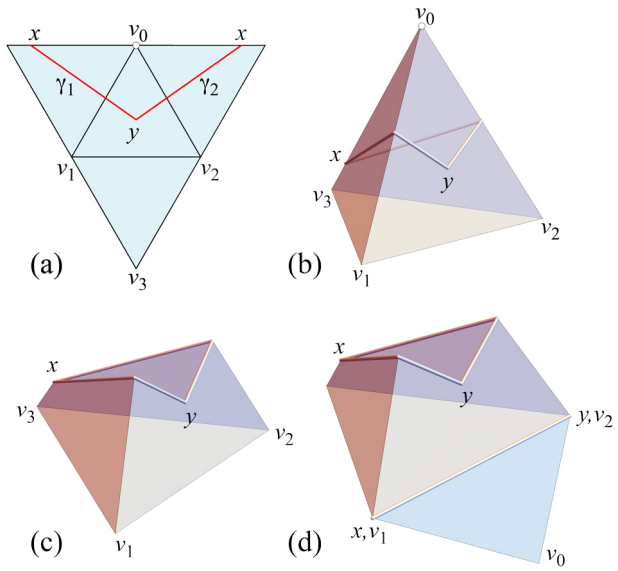


Figure 1: (a) Unfolding of tetrahedron, apex v_0. (b) Digon γ_i connects x to y, surrounding v_0. (c) After removal of the digon. (d) Digon doubly covered triangle sutured along edge v_1v_2.

Cube. Fig. 2 shows excising a unit-cube corner v_0 with geodesics γ_1 and γ_2, each of length 1, and then suturing this digon, folded to a doubly covered triangle, into the edge v_1v_2. After closing the digon hole, the result is a 10-vertex, 16-triangle polyhedron $\mathcal{P}'$.

Doubly Covered Square. Alexandrov's theorem holds for doubly covered, flat convex polygons, and vertex-transplanting does as well. A simple example is cutting off a corner of a doubly covered unit square with a unit length diagonal, and pasting the digon onto another

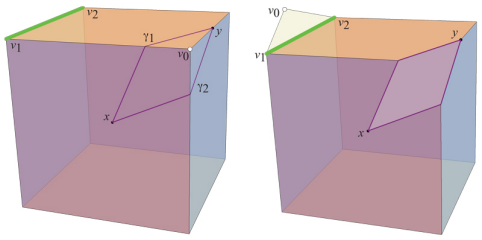


Figure 2: Left: Digon xy surrounding v_0. Right: v_0 transplanted to v_1v_2; v_0 is the apex of a doubly covered triangle, the digon flattened. Hole to be sutured closed to form $\mathcal{P}'$.

edge. The result is another doubly covered polygon: see Fig. 3.

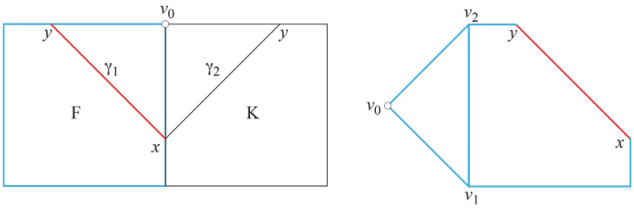


Figure 3: A doubly covered square $\mathcal{P}$ (front F, back K) converted to a doubly covered hexagon $\mathcal{P}'$.

A more interesting example is shown in Fig. 4. The indicated transplant produces a 6-vertex polyhedron $\mathcal{P}'$—combinatorially an octahedron—whose symmetries make exact reconstruction feasible. Vertices v_0 and v_3 retain their curvature π, and the remaining four vertices of $\mathcal{P}'$, v_1, v_2, x, y, each have curvature $\pi/2$.

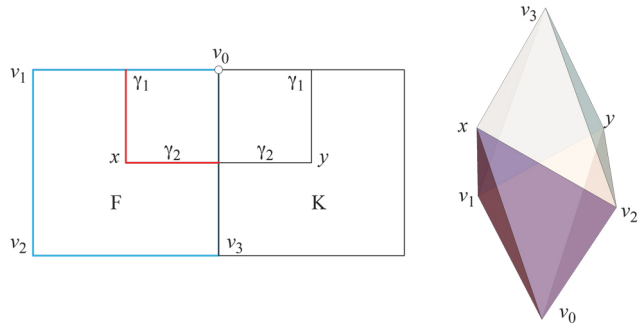


Figure 4: Transplanting v_0 to v_1v_2 on a doubly covered square (front F, back K) leads to a non-flat polyhedron $\mathcal{P}'$.

Doubly Covered Equilateral Triangle. The only polyhedron for which I am certain Theorem 1 (without restrictions) fails is the doubly covered, unit side-length equilateral triangle. The diameter $D = 1$ is realized by the endpoints of any of its three unit-length edges. Any other shortest geodesic is strictly less than 1 in length, as illustrated in Fig. 5. Thus there is no opportunity

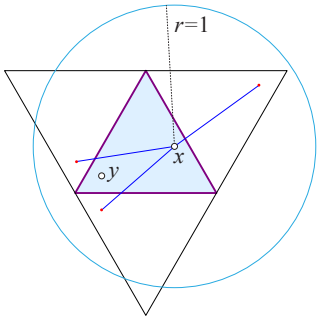


Figure 5: Point x is on the front, y on the back. Three images of y are shown, corresponding to the three paths from x to y. The shortest of these paths is never ≥ 1 unless both x and y are (different) corners.

to create a digon of length 1 surrounding a vertex, but length 1 would be needed to glue into an edge.

3 Preliminaries

Let the vertices of $\mathcal{P}$ be v_i, and let the curvature (angle gap) at v_i be ω_i. We assume all vertices are corners in the sense that $\omega_i > 0$. Let v_0, v_1, v_2 be three vertices, labeled so that ω_0 is smallest among the three, $\leq \omega_1, \omega_2$. Let v_1v_2 be the shortest geodesic on $\mathcal{P}$ connecting v_1 and v_2, with $|v_1v_2| = c$ its length. Often such a shortest geodesic is called a *segment*. A *digon* is a pair of shortest geodesics γ_1, γ_2 of the same length, $|\gamma_1| = |\gamma_2| = c$, connecting two points on $\mathcal{P}$. For us, digons will always surround one vertex v_0. Since shortest paths cannot go through v_0, geodesics slightly left and right of v_0 meet on the "other side" of v_0. We will show that, with careful choice of v_0, v_1, v_2, we can cut out a digon X surrounding v_0, fold it to a doubly covered triangle and paste it into v_1v_2 slit open.

The technique of gluing a triangle along a geodesic v_1v_2 on $\mathcal{P}$ was introduced by [Ale05, p.240], and employed in [OV14] to merge vertices. Excising a digon surrounding a vertex is used in [INV11, Lem. 2]. What seems to be new is excising from one place on $\mathcal{P}$ and inserting elsewhere on $\mathcal{P}$.

Let $C(x)$ be the *cut locus* on $\mathcal{P}$ with respect to point $x \in \mathcal{P}$. (In some computer science literature, this is called the *ridge tree* [AAOS97].) $C(x)$ is the set of points on $\mathcal{P}$ with at least two shortest paths from x. It is a tree composed of shortest paths; in general, each vertex of $\mathcal{P}$ is a degree-1 leaf of the closure of $C(x)$.

We will need to exclude positions of x that are non-generic in that $C(x)$ includes one or more vertices. It was shown in [AAOS97, Lem. 3.8] that the surface of $\mathcal{P}$ may be partitioned into $O(n^4)$ *ridge-free* regions, deter-

mined by overlaying the cut loci of all vertices: $\bigcup_i C(v_i)$. Say that $x \in \mathcal{P}$ is *generic* if it is not a vertex and lies strictly inside a ridge-free region. For later reference, we state this lemma:

Lemma 2 *Within every neighborhood of any point $p \in \mathcal{P}$, there is a generic $x \in P$.*

Proof. This follows because ridge-free regions are bounded by cut-loci arcs, each of which is a 1-dimensional geodesic. □

For generic x, the cut locus in the neighborhood of a vertex v_0 consists of a geodesic segment s open at v_0 and continuing for some positive distance before reaching a junction u of degree-3 or higher. Let $\delta(x,u) = \delta$ be the length of s; see Fig. 6.

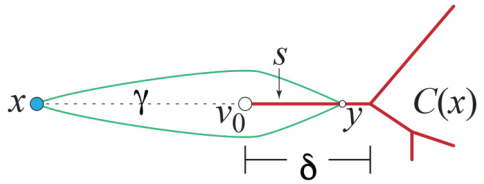


Figure 6: Geodesic segment s of $C(x)$ (red) incident to vertex v_0. A pair of shortest geodesics from x to $y \in s$ are shown (green).

4 Surgery Procedure

We start with and will describe the procedure for any three vertices v_0, v_1, v_2, but later (Sec. 5) we will chose specific vertices.

Let x be a generic point on $\mathcal{P}$ and γ a shortest geodesic to v_0 with length $|\gamma| = |v_1v_2| = c$. The existence of such an x is deferred to Sec. 5. If we move x along γ toward v_0, γ splits into two geodesics γ_1, γ_2 connecting x to a point $y \in C(x)$, with x and y moving in concert while maintaining $|\gamma_1| = |\gamma_2| = c$. If we move x a small enough distance ε, then y will lie on the segment $s \subset C(x)$ as in Fig. 6. Because Lemma 2 allows us to choose x to lie in a ridge-free region R, we can ensure that s has a length $|s| = \delta > 0$. Now γ_1, γ_2 form a digon X surrounding v_0. With sufficiently small ε, we can ensure that X is empty of other vertices, and that y is generic as well. See Fig. 7.

Let the surface angles inside the digon at x and at y be α and β respectively. By Gauss-Bonnet, we have $\alpha + \beta = \omega_0$:

$$\tau + \omega_0 = 2\pi = ((\pi - \alpha) + (\pi - \beta)) + \omega_0 = 2\pi \ ,$$

where the turn angle τ is only non-zero at the endpoints x and y. In particular, $0 < \alpha, \beta < \omega_0$. These inequalities are strict because the digon wraps around v_0 after moving x toward v_0, so $\alpha > 0$.

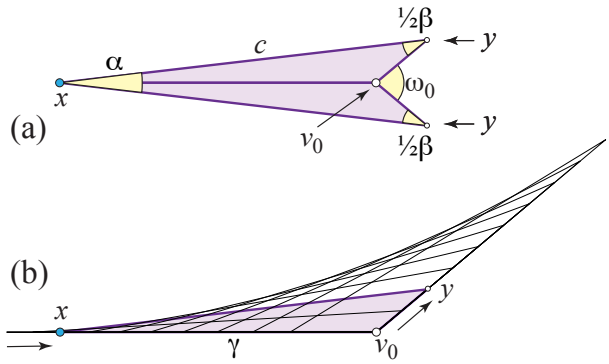


Figure 7: (a) Flattened digon surrounding v_0: $\alpha + \beta = \omega_0$. (b) Sliding x along γ toward v_0, and y along s, while maintaining length c constant.

Now we can suture-in the digon X to a slit along v_1v_2 because:

- The lengths match: $|v_1v_2| = c$ and $|\gamma_1| = |\gamma_2| = c$.
- The curvatures at v_1, v_2 remain positive: $\alpha, \beta < \omega_0 \le \omega_1, \omega_2$, so $\omega_1 - \alpha > 0$ and $\omega_2 - \beta > 0$.

We then close up the digon on the surface of $\mathcal{P}$ and invoke Alexandrov's theorem to obtain $\mathcal{P}'$. We now detail the curvature consequences at the five points involved in the surgery: v_0, v_1, v_2, x, y.

- v_0 is unaltered, just moved, i.e, transplanted.
- Both x and y become vertices after the transplant, of curvatures α and β respectively. Because neither was a vertex (both generic), this accounts for the increase from n to $n+2$ vertices in $\mathcal{P}'$.
- Because $\alpha < \omega_0 \le \omega_1$, the change at v_1 cannot flatten v_1. So v_1 remains a vertex, as does v_2.

We note that the condition that $\omega_0 \le \omega_1, \omega_2$ is in fact more stringent that what is required to ensure that the curvatures at v_1, v_2 remain non-negative. The latter implies that $\omega_0 \le \omega_1 + \omega_2$, a considerably weaker condition. Moreover, our restriction to generic x and y is also not necessary: either or both of x and y could be vertices without obstructing the transplant. Our strict conditions are aimed at guaranteeing a transplant. We leave exploring loosenings to the open problems.

5 Existence of v_0, v_1, v_2

In order to apply the procedure just detailed, we need several conditions to be simultaneously satisfied:

(1) $\omega_0 \le \omega_1, \omega_2$.
(2) $|v_1v_2| = |\gamma_1| = |\gamma_2| = c$.
(3) v_1v_2 should not cross the digon X.

Although (1) is satisfied by any three vertices, just by identifying v_0 with the smallest curvature, the difficulty is that if v_1v_2 is long—say, realizing the diameter of $\mathcal{P}$—then we need there to be an equally long geodesic from x to v_0, to satisfy (2). A solution is to choose v_1 and v_2 to be the nearest neighbors on $\mathcal{P}$, so that $|v_1v_2|$ is small. But then if ω_1, ω_2 are both small, we may not be able to locate a v_0 with a smaller ω_0. We resolve these tensions as follows:

1. We choose v_0 to be a vertex with minimum curvature over all vertices of $\mathcal{P}$, so automatically $\omega_0 \leq \omega_1, \omega_2$ for any choices for v_1 and v_2.
2. Several steps to achieve (2):
   (a) We choose v_1, v_2 to achieve the smallest nearest-neighbor distance $\mathrm{NN}_{\min} = r$ over all pairs of vertices (excluding v_0), so v_1v_2 is as short as possible.
   (b) We prove that the nearest neighbor distance r satisfies $r < \frac{1}{2}D$, where D is the diameter of $\mathcal{P}$.
   (c) We prove that there is an x such that $d(x, v_0) \geq \frac{1}{2}D$.

   Together these imply that we can achieve $|v_1v_2| = |\gamma_1| = |\gamma_2|$.
3. We show that if v_1v_2 crosses X, then another point x may be found that avoids the crossing. This last claim is the only use of the assumption that $\omega_i \leq \pi$ for all vertices v_i.

The next section addresses items (1) and (2) above, and Sec. 7 addresses item (3).

6 Relationship to Diameter D

The *diameter* $D(\mathcal{P})$ of $\mathcal{P}$ is the length of the longest shortest path between any two points. The lemma below ensures we can find a long-enough geodesic $\gamma = xv_0$.

Lemma 3 *For any $x \in \mathcal{P}$, the distance ρ to a point $f(x)$ furthest from x is at least $\frac{1}{2}D$, where $D = D(\mathcal{P})$ is the diameter of $\mathcal{P}$.*

Proof. [2] Let points $y, z \in \mathcal{P}$ realize the diameter: $d(y, z) = D$. For any $x \in \mathcal{P}$,

$$D = d(y, z) \leq d(y, x) + d(x, z)$$

by the triangle inequality on surfaces [Ale06, p.1]. Also we have $\rho \leq d(x, y)$ and $\rho \leq d(x, z)$ because ρ is the furthest distance. So $D = d(y, z) \leq 2\rho$, which establishes the claim. □

[2] Proof suggested by Alexandre Eremenko. `https://mathoverflow.net/a/340056/6094`. See also [IRV19].

Next we establish that the smallest distance (via a shortest geodesic) between a pair of vertices of $\mathcal{P}$, $\mathrm{NN}_{\min}$—the *nearest neighbor distance*—cannot be large with respect to the diameter $D = D(\mathcal{P})$.

6.1 Nearest-Neighbor Distance

Here our goal is to show that sufficiently many points on P cannot all have large nearest-neighbor (NN) distances. First we provide two examples.

1. Let P be a regular tetrahedron with unit edge lengths. D is determined by a point in the center of the base connecting to the apex, so $D = \frac{4}{3}\frac{\sqrt{3}}{2} = \frac{2}{\sqrt{3}}$. The NN-distance is $1 = \frac{\sqrt{3}}{2}D = 0.866\,D$.
2. Let P be a doubly covered regular hexagon, with unit edge lengths. Then $D = 2$, connecting opposite vertices, and the NN-distance is $1 = \frac{1}{2}D$.

Our goal is to ensure the NN distance is at most $\frac{1}{2}D$, which is not achieved by the regular tetrahedron but is for the hexagon. We achieve this by insisting $\mathcal{P}$ have many vertices.

Lemma 4 *Let $\mathcal{P}$ be a polyhedron with diameter D. Let S be a set of distinguished points on $\mathcal{P}$, with $|S| \geq N$. Let r be the smallest* NN-distance *between any pair of points of S. Then $r < D/(\sqrt{N}/2)$. In particular, for $N = 16$, $r < \frac{1}{2}D$.*

Proof.

1. Let a geodesic from x to y realize the diameter D of $\mathcal{P}$. Let U be the *source unfolding* of $\mathcal{P}$ from source point x [DO07, Chap.24.1.1]. U does not self-overlap, and fits inside a circle of radius D; see Fig. 8. Thus the surface area of $\mathcal{P}$ is at most πD^2.
2. Let r be the smallest NN-distance, the smallest separation between a pair of points in S. Then disks of radius $r/2$ centered on points of S have disjoint interiors. For suppose instead two disks overlapped. Then they would be separated by less than r, a contradiction.
3. N non-overlapping disks of radius $r/2$ cover an area of $N\pi(r/2)^2$, which must be less than[3] the surface area of $\mathcal{P}$:

$$N\pi(r/2)^2 < \pi D^2 \quad (1)$$

$$r < \frac{D}{\sqrt{N}/2} \quad (2)$$

   Thus, for $N = 16$, $r < \frac{1}{2}D$.

□

[3] Strictly less than because disk packings leave uncovered gaps.

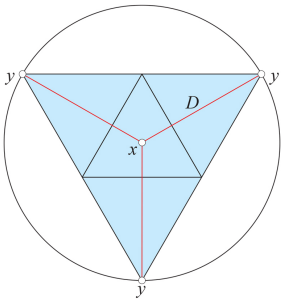


Figure 8: Source unfolding of a regular tetrahedron. xy realizes D.

7 Crossing Avoidance

Although Lemma 3 ensures that we can find an x on the geodesic from $f(v_0)$ to v_0 far enough from v_0 so that we can match $|\gamma|$ with $|v_1v_2|$, if γ crosses v_1v_2, the procedure in Sec. 4 fails. We now detail a method to locate another x in this circumstance. We partition crossings into two cases, long and short.

Recall that v_0 was excluded from the NN calculation of r, so v_0 could be closer to v_1 and/or v_2 than $r = |v_1v_2|$.

Case (1) [long]. Case: $d(v_0, v_i) > r$ for either $i = 1$ or $i = 2$. Assume $d(v_0, v_2) > r$. Then choose $\gamma = v_0v_2$. We can locate x near v_2 on γ to achieve $|xv_0| = r$. See Fig. 9.

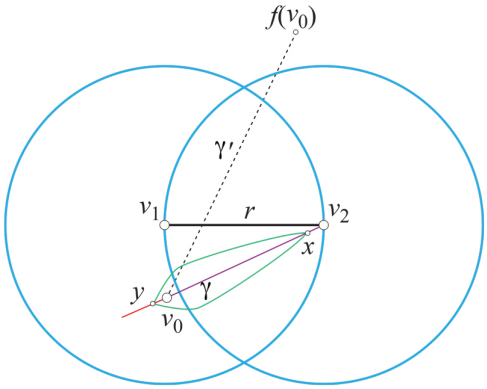


Figure 9: Crossing avoided: $d(v_0, v_2) > r$ (v_0 is outside v_2's r-disk).

Case (2) [short]. If $d(v_0, v_i) \le r$ for $i = 1, 2$, then v_0 is located in the half-lune to the opposite side of (below) v_1v_2 from $f(v_0)$. It is possible that with large curvatures ω_1 and ω_2 that there is no evident "room" below v_1v_2 to locate an x far enough away so that $d(x, v_0) \ge r$. However, with assumptions on the maximum curvature per vertex, room can be found.

The main idea is illustrated in Fig. 10. Although there might not be room either right or left or below for an x achieving $|xv_0| = r$, we can "wrap around" the cone whose apex is v_1 or v_2 to avoid crossing v_1v_2.

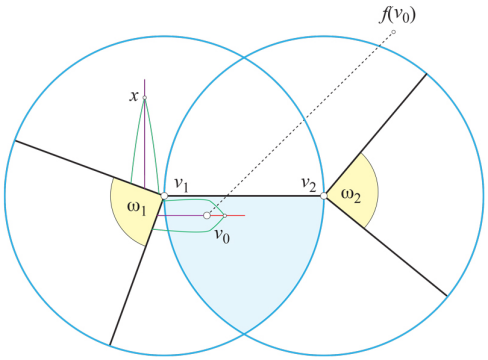


Figure 10: Curvatures $\omega_1 = \omega_2 = \pi/2$. Here xv_0 wraps around v_1.

With larger curvatures at v_1 and v_2, the situation could resemble a doubly covered equilateral triangle with $\omega_i = \frac{4}{3}\pi$, which we saw in Sec. 2 violates Theorem 1. However, if we assume $\omega_i \le \pi$ for all i, a long-enough γ to v_0 can be found.

Assume the worst case, $\omega_1 = \omega_2 = \pi$. As illustrated in Fig. 11, an r-long segment left of v_0 re-enters above v_1v_2 (red), and similarly right of v_0 (green). In fact, it is easy to see that the red and green segments above and below have total length $2r$, regardless of the orientation of the semicircle bounding the angle-gap lines through v_1 and v_2. So there is always enough room to locate x above v_1v_2 connecting "horizontally" to v_0 below.

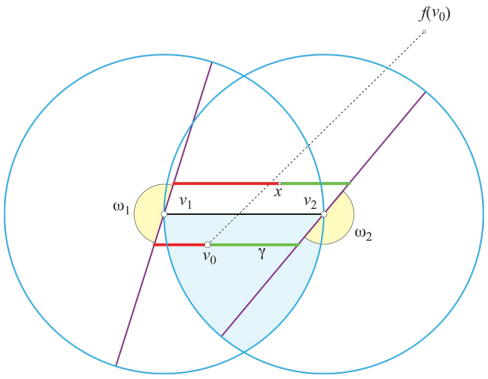


Figure 11: Crossing avoided: Both the red and green segments have total length r each.

8 Open Problems

1. Extend Theorem 1 to all convex polyhedra, i.e., lower $N = 16$ to $N = 4$, and remove the $\omega_i \le \pi$ restriction.

2. Establish conditions that allow more freedom in the selection of the three vertices v_0, v_1, v_2. Right now, Thm. 1 requires following the restrictions detailed in Sec. 5, but as we observed, these restrictions are not necessary for a successful transplant. Additional freedom might permit controlling the shape changes, allowing one to "aim" from $\mathcal{P}$ to some desired $\mathcal{Q}$.

3. Study doubly covered convex polygons as a special case. When does a vertex transplant on a doubly covered polygon produce another doubly covered polygon? See again Sec. 2. (There is a procedure for identifying flat polyhedra [O'R10]; and see [INV11, Lem. 4].)

4. What limit shapes are realized under repeated vertex-transplanting, as $n \to \infty$? Note that because $\alpha, \beta < \omega_0$, new smaller-curvature vertices are created at x and y at each step.

5. Does the transplant guaranteed by Thm. 1 always increase the volume of $\mathcal{P}$? Note that a transplant flattens v_1 and v_2 by α and β, and creates new smallest curvature vertices, $\alpha, \beta < \omega_0$. So the overall effect seems to "round" $\mathcal{P}$.

6. Can Thm. 1 be generalized to transplant several vertices within the same digon? For example, one can excise both endpoints of an edge of a unit cube with a digon of length $\sqrt{2}$ and suture that into a face diagonal.

Related work is under preparation [OV20].

Acknowledgement. I benefitted from the advice of Anna Lubiw and Costin Vîlcu, and suggestions by the referees.

References

[AAOS97] Pankaj K. Agarwal, Boris Aronov, Joseph O'Rourke, and Catherine A. Schevon. Star unfolding of a polytope with applications. *SIAM J. Comput.*, 26:1689–1713, 1997.

[Ale05] Aleksandr D. Alexandrov. *Convex Polyhedra*. Springer-Verlag, Berlin, 2005. Monographs in Mathematics. Translation of the 1950 Russian edition by N. S. Dairbekov, S. S. Kutateladze, and A. B. Sossinsky.

[Ale06] Aleksandr D. Alexandrov. Intrinsic geometry of convex surfaces. In S. S. Kutateladze, editor, *A. D. Alexandrov: Selected Works: Part II*, pages 1–426. Chapman & Hall, Boca Raton, 2006.

[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007. `http://www.gfalop.org`.

[INV11] Jin-ichi Itoh, Chie Nara, and Costin Vîlcu. Continuous flattening of convex polyhedra. In *Centre de Recerca Matemàtica Documents*, volume 8, pages 95–98, 2011. XIV Spanish Meeting on Computational Geometry.

[IRV19] Jin-ichi Itoh, Jöel Rouyer, and Costin Vîlcu. Some inequalities for tetrahedra. `https://arxiv.org/abs/1906.11965`, 2019.

[O'R10] Joseph O'Rourke. On flat polyhedra deriving from Alexandrov's theorem. `http://arxiv.org/abs/1007.2016v2`, July 2010.

[OV14] Joseph O'Rourke and Costin Vîlcu. Development of curves on polyhedra via conical existence. *Comput. Geom.: Theory & Appl.*, 47:149–163, 2014.

[OV20] Joseph O'Rourke and Costin Vîlcu. Tailoring for every body: Reshaping convex polyhedra. In preparation, August 2020.

Fitting a Graph to One-Dimensional Data*

Siu-Wing Cheng† Otfried Cheong‡ Taegyoung Lee†

Abstract

Given n data points in $\mathbb{R}^d$, an appropriate edge-weighted graph connecting the data points finds application in solving clustering, classification, and regresssion problems. The graph proposed by Daitch, Kelner and Spielman (ICML 2009) can be computed by quadratic programming and hence in polynomial time. While a more efficient algorithm would be preferable, replacing quadratic programming is challenging even for the special case of points in one dimension. We develop a dynamic programming algorithm for this case that runs in $O(n^2)$ time.

1 Introduction

Many interesting data sets can be interpreted as point sets in $\mathbb{R}^d$, where the dimension d is the number of features of interest of each data point, and the coordinates are the values of each feature. Given such a data set, graph-based semi-supervised learning is a paradigm for making predictions on the unlabelled data using the proximity among the data points and possibly some labelled data (e.g. [2, 5, 8, 10, 12, 13, 14]). Classification, regression, and clustering are some popular applications. The graph has to be set up first in order to perform the subsequent processing. This requires the determination of the graph edges and the weights to be associated with the edges. For example, let w_{ij} denote the weight determined for the edge that connects two points p_i and p_j, and regression can be performed to predict function values f_i's at the points p_i's by minimizing $\sum_{i,j} w_{ij}(f_i - f_j)^2$, subject to fixing the subset of known f_i's [2]. To allow efficient data analysis, it is important that the weighted graph is sparse.

The graph connectivity should satisfy the property that similar discrete samples are connected. To this end, different proximity graphs have been suggested for connecting proximal points. The *kNN*-graph connects each point to its k nearest neighbors. The ε-ball graph connects each point to all other points that are within a distance ε. After fixing the graph connectivity, edges to "near" points are given large weights and edges to "far away" points are given small weights. That is, the larger the weight of an edge between points p and q, the higher the influence of q on p and vice versa. It is thus inappropriate to use the Euclidean distances among the points as edge weights. Naively setting an edge weight as the reciprocal of the edge length does not work either because the influence of a point is required to fall much more rapidly as that point moves farther away. It has been proposed to associate a weight of $\exp(-\ell^2/2\sigma^2)$ to an edge of Euclidean length ℓ for some *a priori* determined parameter σ (e.g. [10]). A well-tuned σ is important. A slight change in σ may greatly affect the processing outcomes as observed in some previous work (e.g. [12]). Several studies have found the *kNN*-graph and the ε-ball graphs to be inferior to other proximity graphs [2, 3, 13] for which both the graph connectivity and the edge weights are determined simultaneously by solving an optimization problem.

We consider the graph proposed by Daitch, Kelner, and Spielman [2]. It is provably sparse, and experiments have shown that it offers good performance in classification, clustering and regression. This graph is defined via quadratic optimization as follows: Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$. We assign weights $w_{ij} \geqslant 0$ to each pair of points (p_i, p_j), such that $w_{ij} = w_{ji}$ and $w_{ii} = 0$. These weights determine for each point p_i a vector $\vec{v}_i$, as follows:

$$\vec{v}_i = \sum_{j=1}^{n} w_{ij}(p_j - p_i).$$

Let v_i denote $\|\vec{v}_i\|$. The weights are chosen so as to minimize the sum

$$Q = \sum_{i=1}^{n} v_i^2,$$

under the constraint that the weights for each point add up to at least one (to prevent the trivial solution of $w_{ij} = 0$ for all i and j):

$$\sum_{j=1}^{n} w_{ij} \geqslant 1 \qquad \text{for } 1 \leqslant i \leqslant n.$$

*Cheng is supported by Research Grants Council, Hong Kong, China (project no. 16200317). Cheong was supported by ICT R&D program of MSIP/IITP [R0126-15-1108]. Lee received support from both Research Grants Council, Hong Kong, China (project no. 16200317) and ICT R&D program of MSIP/IITP [R0126-15-1108].

†Department of Computer Science and Engineering, HKUST, scheng@cse.ust.hk, tleeaf@cse.ust.hk.

‡Korea Advanced Institute of Science & Technology, otfried@kaist.airpost.net.

The resulting graph contains an edge connecting p_i and p_j if and only if $w_{ij} > 0$.

Daitch et al. [2] showed that there is an optimal solution where at most $(d+1)n$ weights are non-zero. Moreover, in two dimensions, optimal weights can be chosen such that the graph is planar.

The optimal weights can be computed by quadratic programming. A quadratic programming problem with m variables can be solved in $\tilde{O}(m^3)$ time in the worst case [9]. In our case, there are $n(n-1)/2$ variables, which gives a worst-case running time of $\tilde{O}(n^6)$. Graphs based on optimizing other convex quality measures have also been considered [5, 13]. Our goal is to design an algorithm to compute the optimal weights in Daitch et al.'s formulation that is significantly faster than quadratic programming. Perhaps surprisingly, this problem is challenging even for points in one dimension, that is, when all points lie on a line. In this case, it is not difficult to show (Lemma 1) that there is an optimal solution such that $w_{ij} > 0$ if and only if p_i and p_j are consecutive.

Despite its simplicity, the one-dimensional problem can model the task of detecting change points and concept drift in a time series (e.g. [1, 4, 6, 7, 11]); for example, seasonal changes in sales figures and customer behavior. A time series of multi-dimensional data $(z_1, z_2, \cdots)$ is given, and the problem is to decide the time steps t at which there is a "significant change" from z_{t-1} to z_t. Suppose that the "distance" between z_{t-1} and z_t can be computed according to some formula appropriate for the application (e.g. [4]). By forming a path graph with vertices corresponding to the data points and edge weights determined as mentioned previously, one apply clustering algorithms (e.g. [2, 10]) to group "similar" vertices and detect the change points as the boundaries of adjacent clusters. This gives a potential application of the graph fitting problem in one dimension.

In general, although there are only $n-1$ variables in one dimension, the weights in an optimal solution do not seem to follow any simple pattern as we illustrate in the following two examples.

Some weights in an optimal solution can be arbitrarily high. Consider four points p_1, p_2, p_3, p_4 in left-to-right order such that $p_2 - p_1 = p_4 - p_3 = 1$ and $p_3 - p_2 = \varepsilon$. By symmetry, $w_{12} = w_{34}$, and so $v_1 = v_4 = w_{12}$. Since $w_{12} + w_{23} \geqslant 1$ and $w_{23} + w_{34} \geqslant 1$ are trivially satisfied by the requirement that $w_{12} = w_{34} \geqslant 1$, we can make v_2 zero by setting $w_{23} = w_{12}/\varepsilon$. In the optimal solution, $w_{12} = w_{34} = 1$ and $w_{23} = 1/\varepsilon$. So w_{23} can be arbitrarily large.

Given points $p_1, \cdots, p_n$ in left-to-right order, it seems ideal to make v_i a zero vector. One can do this for $i \in [2, n-1]$ by setting $w_{i-1,i}/w_{i,i+1} = (p_{i+1}-p_i)/(p_i-p_{i-1})$, however, some of the constraints $w_i + w_{i+1} \geqslant 1$ may be violated. Even if we are lucky that for $i \in [2, n-1]$, we can set $w_{i-1,i}/w_{i,i+1} = (p_{i+1} - p_i)/(p_i - p_{i-1})$ without violating $w_i + w_{i+1} \geqslant 1$, the solution may not be optimal as we show below. Requiring $v_i = 0$ for $i \in [2, n-1]$ gives $v_1 = v_n = w_{12}(p_2 - p_1)$. In general, we have $p_2 - p_1 \neq p_n - p_{n-1}$, so we can assume that $p_2 - p_1 > p_n - p_{n-1}$. Then, $w_{n-1,n} = w_{12}(p_2 - p_1)/(p_n - p_{n-1}) > 1$ as $w_{12} \geqslant 1$. Since $w_{n-1,n} > 1$, one can decrease $w_{n-1,n}$ by a small quantity δ while keeping its value greater than 1. Both constraints $w_{n-1,n} \geqslant 1$ and $w_{n-2,n-1} + w_{n-1,n} \geqslant 1$ are still satisfied. Observe that v_n drops to $w_{12}(p_2 - p_1) - \delta(p_n - p_{n-1})$ and v_{n-1} increases to $\delta(p_n - p_{n-1})$. Hence, $v_{n-1}^2 + v_n^2$ decreases by $2\delta w_{12}(p_2 - p_1)(p_n - p_{n-1}) - 2\delta^2(p_n - p_{n-1})^2$, and so does Q. The original setting of the weights is thus not optimal. If $w_{n-3,n-2} + w_{n-2,n-1} > 1$, it will bring further benefit to decrease $w_{n-2,n-1}$ slightly so that v_{n-1} decreases slightly from $\delta(p_n - p_{n-1})$ and v_{n-2} increases slightly from zero. Intuitively, instead of concentrating $w_{12}(p_2 - p_1)$ at v_n, it is better to distribute it over multiple points in order to decrease the sum of squares. But it does not seem easy to determine the best weights.

Although there are only $n-1$ variables in one dimension, quadratic programming still yields a running time of $\tilde{O}(n^3)$. We present a dynamic programming algorithm that computes the optimal weights in $O(n^2)$ time in the one-dimensional case. The intermediate solution has an interesting structure such that the derivative of its quality measure depends on the derivative of a subproblem's quality measure as well as the inverse of this derivative function. This makes it unclear how to bound the size of an explicit representation of the intermediate solution. Instead, we develop an implicit representation that facilitates the dynamic programming algorithm.

2 A single-parameter quality measure function

We will assume that the points are given in sorted order, so that $p_1 < p_2 < p_3 < \cdots < p_n$. We first argue that the only weights that need to be non-zero are the weights between consecutive points, that is, weights of the form $w_{i,i+1}$.

Lemma 1 *For $d = 1$, there is an optimal solution where only weights between consecutive points are non-zero.*

Proof. Let the width of an optimal solution S be $\max\{|i-k| : w_{ik} > 0 \text{ in } S\}$. Among all optimal solutions, consider the solution O with the minimum width, and in case of ties, pick O to minimize the number of non-zero weights that achieve the minimum width.

Assume to the contrary that the width of O is at least two, achieved by $w_{ik} > 0$ for some $i < k-1$. Let j be an arbitrary index strictly between i and k. We construct a new optimal solution as follows: Let $a = p_j - p_i$, $b = p_k - p_j$, and $w = w_{ik}$. In the new solution, we set $w_{ik} = 0$, increase w_{ij} by $\frac{a+b}{a}w$, and increase w_{jk} by $\frac{a+b}{b}w$. Note that since $a + b > a$ and $a + b > b$,

the sum of weights at each vertex increases, and so the weight vector remains feasible. The value v_j changes by $-a \times \frac{a+b}{a} w + b \times \frac{a+b}{b} w = 0$, the value v_i changes by $-(a+b) \times w + a \times \frac{a+b}{a} w = 0$, and the value v_k changes by $+(a+b) \times w - b \times \frac{a+b}{b} w = 0$. It follows that the new solution has the same quality as the original one, and is therefore also optimal. But then we should have preferred this optimal solution to O, a contradiction. □

To simplify the notation, we set $d_i = p_{i+1} - p_i$, for $1 \leqslant i < n$, rename the weights as $w_i := w_{i,i+1}$, again for $1 \leqslant i < n$, and observe that

$$\begin{aligned} v_1 &= w_1 d_1, \\ v_i &= |w_i d_i - w_{i-1} d_{i-1}| \qquad \text{for } 2 \leqslant i \leqslant n-1, \\ v_n &= w_{n-1} d_{n-1}. \end{aligned}$$

For $i \in [2, n-1]$, we introduce the quantity

$$\begin{aligned} Q_i &= d_i^2 w_i^2 + \sum_{j=1}^{i} v_j^2 \\ &= d_i^2 w_i^2 + d_1^2 w_1^2 + \sum_{j=2}^{i} (d_j w_j - d_{j-1} w_{j-1})^2, \end{aligned}$$

and note that $Q_{n-1} = \sum_{i=1}^{n} v_i^2 = Q$. Thus, our goal is to choose the $n-1$ non-negative weights $w_1, \ldots, w_{n-1}$ such that Q_{n-1} is minimized, under the constraints

$$\begin{aligned} w_1 &\geqslant 1, \\ w_j + w_{j+1} &\geqslant 1 \qquad \text{for } 2 \leqslant j \leqslant n-2, \\ w_{n-1} &\geqslant 1. \end{aligned}$$

The quantity Q_i depends on $w_1, w_2, \ldots, w_i$. We concentrate on w_i and consider the function

$$w_i \mapsto Q_i(w_i) = \min_{w_1, \ldots, w_{i-1}} Q_i(w_1, w_2, \ldots, w_{i-1}, w_i),$$

where the minimum is taken over all choices of $w_1, \ldots, w_{i-1}$ that respect the constraints $w_1 \geqslant 1$ and $w_j + w_{j+1} \geqslant 1$ for $2 \leqslant j \leqslant i-1$. The function $Q_i(w_i)$ is defined on $[0, \infty)$.

We denote the derivative of the function $w_i \mapsto Q_i(w_i)$ by R_i. We will see shortly that R_i is a continuous, piecewise linear function. Since R_i is not differentiable everywhere, we define $S_i(x)$ to be the right derivative of R_i, that is

$$S_i(x) = \lim_{y \to x^+} R_i'(y).$$

The following result discusses R_i and S_i. The shorthand

$$\xi_i := 2 d_i d_{i+1}, \quad \text{for } 1 \leqslant i < n-1,$$

will be convenient in its proof and the rest of the paper.

Theorem 2 *The function R_i is strictly increasing, continuous, and piecewise linear on the range $[0, \infty)$. We have $R_i(0) < 0$, $S_i(x) \geqslant (2 + 2/i) d_i^2$ for all $x \geqslant 0$, and $R_i(x) = (2 + 2/i) d_i^2 x$ for sufficiently large $x > 0$.*

Proof. We prove all claims by induction over i. The base case is $i = 2$. Observe that

$$Q_2 = v_1^2 + v_2^2 + d_2^2 w_2^2 = 2 d_1^2 w_1^2 - 2 d_1 d_2 w_1 w_2 + 2 d_2^2 w_2^2.$$

The derivative with respect to w_1 is

$$\frac{\partial}{\partial w_1} Q_2 = 4 d_1^2 w_1 - 2 d_1 d_2 w_2, \tag{1}$$

which implies that Q_2 is minimized for $w_1 = \frac{d_2}{2 d_1} w_2$. This choice is feasible (with respect to the constraint $w_1 \geqslant 1$) when $w_2 \geqslant \frac{2 d_1}{d_2}$. If $w_2 < \frac{2 d_1}{d_2}$, then $\frac{\partial}{\partial w_1} Q_2$ is positive for all values of $w_1 \geqslant 1$, so the minimum occurs at $w_1 = 1$. It follows that

$$Q_2(w_2) = \begin{cases} \frac{3}{2} d_2^2 w_2^2 & \text{for } w_2 \geqslant \frac{2 d_1}{d_2}, \\ 2 d_2^2 w_2^2 - \xi_1 w_2 + 2 d_1^2 & \text{otherwise,} \end{cases}$$

and so we have

$$R_2(w_2) = \begin{cases} 3 d_2^2 w_2 & \text{for } w_2 \geqslant \frac{2 d_1}{d_2}, \\ 4 d_2^2 w_2 - \xi_1 & \text{otherwise.} \end{cases} \tag{2}$$

In other words, R_2 is piecewise linear and has a single breakpoint at $\frac{2 d_1}{d_2}$. The function R_2 is continuous because $3 d_2^2 w_2 = 4 d_2^2 w_2 - \xi_1$ when $w_2 = \frac{2 d_1}{d_2}$. We have $R_2(0) = -\xi_1 < 0$, $S_2(x) \geqslant 3 d_2^2$ for all $x \geqslant 0$, and $R_2(x) = 3 d_2^2 x$ for $x \geqslant \frac{2 d_1}{d_2}$. The fact that $S_2(x) \geqslant 3 d_2^2 > 0$ makes R_2 strictly increasing.

Consider now $i \geqslant 2$, assume that R_i and S_i satisfy the induction hypothesis, and consider Q_{i+1}. By definition, we have

$$Q_{i+1} = Q_i - \xi_i w_i w_{i+1} + 2 d_{i+1}^2 w_{i+1}^2. \tag{3}$$

For a given value of $w_{i+1} \geqslant 0$, we need to find the value of w_i that will minimize Q_{i+1}. The derivative is

$$\frac{\partial}{\partial w_i} Q_{i+1} = R_i(w_i) - \xi_i w_{i+1}.$$

The minimum thus occurs when $R_i(w_i) = \xi_i w_{i+1}$.

Since R_i is a strictly increasing continuous function with $R_i(0) < 0$ and $\lim_{x \to \infty} R_i(x) = \infty$, for any given $w_{i+1} \geqslant 0$, there exists a unique value $w_i = R_i^{-1}(\xi_i w_{i+1})$. However, we also need to satisfy the constraint $w_i + w_{i+1} \geqslant 1$.

We first show that R_{i+1} is continuous and piecewise linear, and that $R_{i+1}(0) < 0$. We will distinguish two cases, based on the value of $w_i^\circ := R_i^{-1}(0)$.

Case 1: $w_i^\circ \geqslant 1$. This means that $R_i^{-1}(\xi_i w_{i+1}) \geqslant 1$ for any $w_{i+1} \geqslant 0$, and so the constraint of $w_i + w_{i+1} \geqslant 1$ is satisfied for the optimal choice of $w_i = R_i^{-1}(\xi_i w_{i+1})$. It follows that

$$\begin{aligned} Q_{i+1}(w_{i+1}) = Q_i\big(R_i^{-1}(\xi_i w_{i+1})\big) - \xi_i w_{i+1} R_i^{-1}(\xi_i w_{i+1}) \\ + 2d_{i+1}^2 w_{i+1}^2. \end{aligned}$$

The derivative R_{i+1} is therefore

$$\begin{aligned} R_{i+1}(w_{i+1}) &= R_i(R_i^{-1}(\xi_i w_{i+1}))\frac{\xi_i}{R_i'(R_i^{-1}(\xi_i w_{i+1}))} \\ &\quad - \xi_i R_i^{-1}(\xi_i w_{i+1}) \\ &\quad - \xi_i w_{i+1}\frac{\xi_i}{R_i'(R_i^{-1}(\xi_i w_{i+1}))} \\ &\quad + 4d_{i+1}^2 w_{i+1} \\ &= 4d_{i+1}^2 w_{i+1} - \xi_i R_i^{-1}(\xi_i w_{i+1}). \end{aligned} \tag{4}$$

Since R_i is continuous and piecewise linear, so is R_i^{-1}, and therefore R_{i+1} is continuous and piecewise linear. We have $R_{i+1}(0) = -\xi_i w_i^\circ < 0$.

Case 2: $w_i^\circ < 1$. Consider the function $x \mapsto f(x) = x + R_i(x)/\xi_i$. Since R_i is continuous and strictly increasing by the inductive assumption, so is the function f. Observe that $f(w_i^\circ) = w_i^\circ < 1$. As $w_i^\circ < 1$, we have $R_i(1) > R_i(w_i^\circ) = 0$, which implies that $f(1) > 1$. Thus, there exists a unique value $w_i^\bowtie \in (w_i^\circ, 1)$ such that $f(w_i^\bowtie) = w_i^\bowtie + R_i(w_i^\bowtie)/\xi_i = 1$.

For $w_{i+1} \geqslant 1 - w_i^\bowtie = R_i(w_i^\bowtie)/\xi_i$, we have $R_i^{-1}(\xi_i w_{i+1}) \geqslant w_i^\bowtie$, and so $R_i^{-1}(\xi_i w_{i+1}) + w_{i+1} \geqslant 1$. This implies that the constraint $w_i + w_{i+1} \geqslant 1$ is satisfied when $Q_{i+1}(w_{i+1})$ is minimized for the optimal choice of $w_i = R_i^{-1}(\xi_i w_{i+1})$. So R_{i+1} is as in (4) in Case 1.

When $w_{i+1} < 1 - w_i^\bowtie$, the constraint $w_i + w_{i+1} \geqslant 1$ implies that $w_i \geqslant 1 - w_{i+1} > w_i^\bowtie$. For any $w_i > w_i^\bowtie$ we have $\frac{\partial}{\partial w_i} Q_{i+1} = R_i(w_i) - \xi_i w_{i+1} > R_i(w_i^\bowtie) - \xi_i(1 - w_i^\bowtie) = 0$. So Q_{i+1} is increasing, and the minimal value is obtained for the smallest feasible choice of w_i, that is, for $w_i = 1 - w_{i+1}$. It follows that

$$\begin{aligned} Q_{i+1}(w_{i+1}) &= Q_i(1 - w_{i+1}) - \xi_i w_{i+1}(1 - w_{i+1}) \\ &\quad + 2d_{i+1}^2 w_{i+1}^2 \\ &= Q_i(1 - w_{i+1}) - \xi_i w_{i+1} \\ &\quad + (\xi_i + 2d_{i+1}^2) w_{i+1}^2, \end{aligned}$$

and so the derivative R_{i+1} is

$$\begin{aligned} R_{i+1}(w_{i+1}) = -R_i(1 - w_{i+1}) \\ + (2\xi_i + 4d_{i+1}^2) w_{i+1} - \xi_i. \end{aligned} \tag{5}$$

Combining (4) and (5), we have

- If $w_{i+1} < 1 - w_i^\bowtie$, then

$$\begin{aligned} R_{i+1}(w_{i+1}) = -R_i(1 - w_{i+1}) \\ + (2\xi_i + 4d_{i+1}^2) w_{i+1} - \xi_i. \end{aligned} \tag{6}$$

- If $w_{i+1} \geqslant 1 - w_i^\bowtie$, then

$$R_{i+1}(w_{i+1}) = 4d_{i+1}^2 w_{i+1} - \xi_i R_i^{-1}(\xi_i w_{i+1}). \tag{7}$$

For $w_{i+1} = 1 - w_i^\bowtie$, we have $R_i(1 - w_{i+1}) = R_i(w_i^\bowtie) = \xi_i(1 - w_i^\bowtie)$ and $R_i^{-1}(\xi_i w_{i+1}) = R_i^{-1}(\xi_i(1 - w_i^\bowtie)) = w_i^\bowtie$, and so both expressions have the same value:

$$\begin{aligned} &- R_i(1 - w_{i+1}) + (2\xi_i + 4d_{i+1}^2) w_{i+1} - \xi_i \\ &= \xi_i w_i^\bowtie - \xi_i + 2\xi_i - 2\xi_i w_i^\bowtie + 4d_{i+1}^2(1 - w_i^\bowtie) - \xi_i \\ &= 4d_{i+1}^2(1 - w_i^\bowtie) - \xi_i w_i^\bowtie \\ &= 4d_{i+1}^2(1 - w_i^\bowtie) - \xi_i R_i^{-1}(\xi_i w_{i+1}). \end{aligned}$$

Since R_i is continuous and piecewise linear, this implies that R_{i+1} is continuous and piecewise linear. We have $R_{i+1}(0) = -R_i(1) - \xi_i$. Since $w_i^\circ < 1$, we have $R_i(1) > R_i(w_i^\circ) = 0$, and so $R_{i+1}(0) < 0$.

Next, we show that $S_{i+1}(x) \geqslant (2 + 2/i+1) d_{i+1}^2$ for all $x \geqslant 0$, which implies that R_{i+1} is strictly increasing. If $w_i^\circ < 1$ and $x < 1 - w_i^\bowtie$, then by (6),

$$\begin{aligned} S_{i+1}(x) &= S_i(1 - x) + 2\xi_i + 4d_{i+1}^2 \\ &> 4d_{i+1}^2 \\ &> (2 + 2/i+1) d_{i+1}^2. \end{aligned}$$

If $w_i^\circ \geqslant 1$ or $x > 1 - w_i^\bowtie$, we have by (4)and (7) that $R_{i+1}(x) = 4d_{i+1}^2 x - \xi_i R_i^{-1}(\xi_i x)$. By the inductive assumption that $S_i(x) \geqslant (2 + 2/i) d_i^2$ for all $x \geqslant 0$, we get $\frac{\partial}{\partial x} R_i^{-1}(x) \leqslant 1/\big((2 + 2/i) d_i^2\big)$. It follows that

$$\begin{aligned} S_{i+1}(x) \geqslant\ 4d_{i+1}^2 - \frac{(2d_i d_{i+1})^2}{(2 + 2/i) d_i^2} &= \Big(4 - \frac{4}{2 + 2/i}\Big) d_{i+1}^2 \\ &= \Big(4 - \frac{2i}{i+1}\Big) d_{i+1}^2 \\ &= \Big(2 + \frac{2}{i+1}\Big) d_{i+1}^2. \end{aligned}$$

This establishes the lower bound on $S_{i+1}(x)$.

Finally, by the inductive assumption, when x is large enough, we have $R_i^{-1}(x) = x/\big((2 + 2/i) d_i^2\big)$, and so

$$\begin{aligned} R_{i+1}(x) &= 4d_{i+1}^2 x - \frac{(2d_i d_{i+1})^2}{(2 + 2/i) d_i^2} x \\ &= \Big(2 + \frac{2}{i+1}\Big) d_{i+1}^2 x, \end{aligned}$$

completing the inductive step and therefore the proof. □

3 The algorithm

Our algorithm progressively constructs a representation of the functions $R_2, R_3, \ldots, R_{n-1}$. The function representation supports the following three operations:

- Op 1: given x, return $R_i(x)$;
- Op 2: given y, return $R_i^{-1}(y)$;
- Op 3: given ξ, return $x^{\bowtie}$ such that $x^{\bowtie} + \frac{R_i(x^{\bowtie})}{\xi} = 1$.

The proof of Theorem 2 gives the relation between R_{i+1} and R_i. This will allow us to construct the functions one by one—we discuss the detailed implementation in Sections 3.1 and 3.2 below.

Once all functions $R_2, \ldots, R_{n-1}$ are constructed, the optimal weights $w_1, w_2, \ldots, w_{n-1}$ are computed from the R_i's as follows. Recall that $Q = Q_{n-1}$, so w_{n-1} is the value minimizing $Q_{n-1}(w_{n-1})$ under the constraint $w_{n-1} \geqslant 1$. If $R_{n-1}^{-1}(0) \geqslant 1$, then $R_{n-1}^{-1}(0)$ is the optimal value for w_{n-1}; otherwise, we set w_{n-1} to 1.

To obtain w_{n-2}, recall from (3) that $Q = Q_{n-1} = Q_{n-2}(w_{n-2}) - \xi_{n-2}w_{n-2}w_{n-1} + 2d_{n-1}^2 w_{n-1}^2$. Since we have already determined the correct value of w_{n-1}, it remains to choose w_{n-2} so that Q_{n-1} is minimized. Since

$$\frac{\partial}{\partial w_{n-2}} Q_{n-1} = R_{n-2}(w_{n-2}) - \xi_{n-2}w_{n-1},$$

Q_{n-1} is minimized when $R_{n-2}(w_{n-2}) = \xi_{n-2}w_{n-1}$, and so $w_{n-2} = R_{n-2}^{-1}(\xi_{n-2}w_{n-1})$.

In general, for $i \in [2, n-2]$, we can obtain w_i from w_{i+1} by observing that

$$Q_{n-1} = Q_i(w_i) - \xi_i w_i w_{i+1} + g(w_{i+1}, \ldots, w_{n-1}),$$

where g is function that only depends on $w_{i+1}, \ldots, w_{n-1}$. Taking the derivative again, we have

$$\frac{\partial}{\partial w_i} Q_{n-1} = R_i(w_i) - \xi_i w_{i+1},$$

so choosing $w_i = R_i^{-1}(\xi_i w_{i+1})$ minimizes Q_{n-1}. To also satisfy the constraint $w_i + w_{i+1} \geqslant 1$, we need to choose $w_i = \max\{R_i^{-1}(\xi_i w_{i+1}),\, 1 - w_{i+1}\}$ for $i \in [2, n-2]$. Finally, from the discussion that immediately follows (1), we set $w_1 = \max\{\frac{d_2}{2d_1} w_2,\, 1\}$. To summarize, we have

$$\begin{aligned} w_{n-1} &= \max\{R_{n-1}^{-1}(0),\, 1\}, \\ w_i &= \max\{R_i^{-1}(\xi_i w_{i+1}),\, 1 - w_{i+1}\}, \text{ for } i \in [2, n-2], \\ w_1 &= \max\{\tfrac{d_2}{2d_1} w_2,\, 1\}. \end{aligned}$$

It follows that we can obtain the optimal weights using a single Op 2 on each R_i.

3.1 Explicit representation of piecewise linear functions

Since R_i is a piecewise linear function, a natural representation is a sequence of linear functions, together with the sequence of breakpoints. Since R_i is strictly increasing, all three operations can then be implemented to run in time $O(\log k)$ using binary search, where k is the number of function pieces.

We construct the functions R_i, for $i = 2, \ldots, n-1$, one by one.

The function R_2 consists of exactly two pieces. We construct it directly from d_1, d_2, and ξ_1 using (2).

To construct R_{i+1}, we make use of the explicit representation of R_i that we have already computed. We first compute $w_i^{\circ} = R_i^{-1}(0)$ using Op 2 on R_i. If $w_i^{\circ} \geqslant 1$, then by (4) each piece of R_i, starting at the x-coordinate w_i°, gives rise to a linear piece of R_{i+1}, so the number of pieces of R_{i+1} is at most that of R_i.

If $w_i^{\circ} < 1$, then we compute $w_i^{\bowtie}$ using Op 3 on R_i. The new function R_{i+1} has a breakpoint at $1 - w_i^{\bowtie}$ by (6) and (7). Its pieces for $x \geqslant 1 - w_i^{\bowtie}$ are computed from the pieces of R_i starting at the x-coordinate $w_i^{\bowtie}$. Its pieces for $0 \leqslant x < 1 - w_i^{\bowtie}$ are computed from the pieces of R_i between the x-coordinates 1 and $w_i^{\bowtie}$. (Increasing w_{i+1} now corresponds to a decreasing w_i.) This implies that every piece of R_i that covers x-coordinates in the range $[w_i^{\bowtie}, 1]$ will give rise to *two* pieces of R_{i+1}, so the number of pieces of R_{i+1} may be twice the number of pieces of R_i.

Therefore, although this method works, it is unclear whether the number of linear pieces of R_i is bounded by a polynomial in i.

3.2 A quadratic time implementation

Since we have no polynomial bound on the number of linear pieces of the function R_{n-1}, we turn to an implicit representation of R_i.

The representation is based on the fact that there is a linear relationship between points on the graphs of the functions R_i and R_{i+1}. Concretely, let $y_i = R_i(x_i)$, and $y_{i+1} = R_{i+1}(x_{i+1})$. Recall the following relation from (4) for the case of $w_i^{\circ} \geqslant 1$:

$$R_{i+1}(w_{i+1}) = 4d_{i+1}^2 w_{i+1} - \xi_i R_i^{-1}(\xi_i w_{i+1}).$$

We can express this relation as a system of two equations:

$$\begin{aligned} y_{i+1} &= 4d_{i+1}^2 x_{i+1} - \xi_i x_i, \\ y_i &= \xi_i x_{i+1}. \end{aligned}$$

This can be rewritten as

$$\begin{aligned} y_{i+1} &= 4d_{i+1}^2 y_i/\xi_i - \xi_i x_i, \\ x_{i+1} &= y_i/\xi_i, \end{aligned}$$

or in matrix notation

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \\ 1 \end{pmatrix} = M_{i+1} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}, \tag{8}$$

where

$$M_{i+1} = \begin{pmatrix} 0 & 1/\xi_i & 0 \\ -\xi_i & 4d_{i+1}^2/\xi_i & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

On the other hand, if $w_i^\circ < 1$, then R_{i+1} has a breakpoint at $1 - w_i^\bowtie$. The value $w_i^\bowtie$ can be obtained by appying Op 3 to R_i. We compute the coordinates of this breakpoint: $(1 - w_i^\bowtie, R_{i+1}(1 - w_i^\bowtie))$. Note that $R_{i+1}(1 - w_i^\bowtie) = 4d_{i+1}^2(1 - w_i^\bowtie) - \xi_i R_i^{-1}(\xi_i(1 - w_i^\bowtie))$ which can be computed by applying Op 2 to R_i. For $x_{i+1} > 1 - w_i^\bowtie$, the relationship between (x_i, y_i) and (x_{i+1}, y_{i+1}) is given by (8). For $0 \leqslant x_{i+1} < 1 - w_i^\bowtie$, recall from (5) that

$$\begin{aligned} R_{i+1}(w_{i+1}) = &-R_i(1 - w_{i+1}) \\ &+ (2\xi_i + 4d_{i+1}^2)w_{i+1} - \xi_i. \end{aligned}$$

We again rewrite this as

$$\begin{aligned} y_{i+1} &= -y_i + (2\xi_i + 4d_{i+1}^2)x_{i+1} - \xi_i, \\ x_i &= 1 - x_{i+1}, \end{aligned}$$

which gives

$$\begin{aligned} y_{i+1} &= -y_i + (2\xi_i + 4d_{i+1}^2)(1 - x_i) - \xi_i, \\ x_{i+1} &= 1 - x_i, \end{aligned}$$

or in matrix notation:

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \\ 1 \end{pmatrix} = L_{i+1} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix},$$

where

$$L_{i+1} = \begin{pmatrix} -1 & 0 & 1 \\ -2\xi_i - 4d_{i+1}^2 & -1 & \xi_i + 4d_{i+1}^2 \\ 0 & 0 & 1 \end{pmatrix}.$$

We will make use of this relationship to store the function R_{i+1}, for $i \geqslant 2$, by storing the breakpoint $(x_{i+1}^*, y_{i+1}^*) = (1 - w_i^\bowtie, R_{i+1}(1 - w_i^\bowtie))$ as well as the two matrices L_{i+1} and M_{i+1}. The function R_2 is simply stored explicitly.

We now discuss how the three operations Op 1, Op 2, and Op 3 are implemented on this representation of a function R_i. For an operation on R_i, we progressively build transformation matrices $T_i^i, T_{i-1}^i, T_{i-2}^i, \ldots, T_3^i, T_2^i$ such that $(x_i, y_i, 1) = T_j^i \times (x_j, y_j, 1)$ for every $2 \leqslant j \leqslant i$ in a neighborhood of the query. Once we obtain T_2^i, we use our explicit representation of R_2 to express y_i as a linear function of x_i in a neighborhood of the query, which then allows us to answer the query.

The first matrix T_i^i is the identity matrix. We obtain T_j^i from T_{j+1}^i, for $j \in [2, i-1]$, as follows: If R_{j+1} has no breakpoint, then $T_j^i = T_{j+1}^i \cdot M_{j+1}$. If R_{j+1} has a breakpoint (x_{j+1}^*, y_{j+1}^*), then either $T_j^i = T_{j+1}^i \cdot M_{j+1}$ or $T_j^i = T_{j+1}^i \cdot L_{j+1}$, depending on which side of the breakpoint applies to the answer of the query. We can decide this by comparing $(x', y', 1)^t = T_{j+1}^i \cdot (x_{j+1}^*, y_{j+1}^*, 1)^t$ with the query. More precisely, for Op 1 we compare the input x with x', for Op 2 we compare the input y with y', and for Op 3 we compute $x' + y'/\xi$ and compare with 1.

Assuming the Real-RAM model common in computational geometry, where arithmetic on real numbers takes constant time, it follows that the implicit representation of R_i supports all three operations on R_i in time $O(i)$.

Finally, we discuss how the representation of all functions R_i is obtained. We again build it iteratively, constructing $R_2, R_3, R_4, \ldots, R_{n-1}$, one-by-one in this order. The first function R_2 is stored explicitly. To construct the implicit representation of R_{i+1}, we only need to perform on our representation of R_i (that we already computed) one Op 2 to get $w_i^\circ = R_i^{-1}(0)$, one Op 3 to get $w_i^\bowtie$, and one Op 2 to get $R_i^{-1}(\xi_i(1 - w_i^\bowtie))$, which allows us to determine the breakpoint $(1 - w_i^\bowtie, R_{i+1}(1 - w_i^\bowtie))$, if there is one (when $w_i^\circ < 1$). The two matrices L_{i+1} and R_{i+1} can be computed in $O(1)$ time.

Since operations on R_i take time $O(i)$, the total time to construct R_{n-1} is $O(n^2)$.

Theorem 3 *Given n points on a line, we can compute an optimal set of weights for minimizing the quality measure Q in $O(n^2)$ time under the Real-RAM model.*

4 Conclusion

We do not have a polynomial time bound on the running time using the explicit representation of the functions R_i. Future work should determine if there is good bound on the number of pieces in the explicit representation, or an example in which the number of pieces is large.

It would also be nice to obtain an algorithm for higher dimensions that is not based on a quadratic programming solver.

In two dimensions, we have conducted some experiments that indicate that the Delaunay triangulation of the point set contains a well-fitting graph. If we choose the graph edges only from the Delaunay edges and compute the optimal edge weights, the resulting quality measure is very close to the best quality measure in the unrestricted case. It is conceivable that one can obtain a provably good approximation from the Delaunay triangulation.

References

[1] S. Aminikhanghahi and D.J. Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51 (2017), 339–367.

[2] S.I. Daitch, J.A. Kelner, and D.A. Spielman. Fitting a graph to vector data. *Proceedings of the 26th International conference on Machine Learing*, 2009, 201–208.

[3] S. Han, H. Huang, H. Qin, and D. Yu. Locality-preserving L1-graph and its application in clustering. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, 813–818.

[4] S. Hido, T. Idé, H. Kahsima, H. Kubo, and H. Matsuzawa. Unsupervised change analysis using supervised learning. *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2008, 148–159.

[5] T. Jebara, J. Wang, and S.-F. Chang. Graph construction and b-matching for semi-supervised learning. *Proceedings of the 26th International conference on Machine Learing*, 2009, 441–448.

[6] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. *Proceedings of the 17th International Conference on Machine Learning*, 2000, 487–494.

[7] W.-H. Lee, J. Ortiz, B. Ko, and R. Lee. Time series segmentation through automatic feature learning. arXiv:1801.05394v2, 2018.

[8] W. Liu, J. He, and S.-F. Chang. Large graph construction for scalable semi-supervised learning. *Proceedings of the 27th International Conference on Machine Learning*, 2010, 679–686.

[9] R.D.C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part II: convex quadratic programming. *Mathematical Programming*, 44:43–66, 1989.

[10] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: analysis and an algorithm. *Proceedings of the 14th International Conference on Neural Information Processing Systems*, 2001, 849–856.

[11] M. Scholz and R. Klinkenberg. Boosting classfiers for drifting concepts. *Intelligent Data Analysis - Knowledge Discovery from Data Streams*, 11 (2007), 3–28.

[12] S. Xiang, F. Nie, and C. Zhang. Semi-supervised classification via local spline regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32 (2010), 2039–2053.

[13] Y.-M. Zhang, K. Huang, and C.-L. Liu. Learning locality preserving graph from data. *IEEE Transactions on Cybernetics*, 44 (2014), 2088–2098.

[14] D. Zhou, O. Bousquet, T.N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. *Proceedings of the 16th International Conference on Neural Information Processing Systems*, 2003, 321–328.

Planar Emulators for Monge Matrices

Hsien-Chih Chang* Tim Ophelders†

Abstract

We constructively show that any cyclic Monge distance matrix can be represented as the graph distances between vertices on the outer face of a planar graph. The structure of the planar graph depends only on the number of rows of the matrix, and the weight of each edge is a fixed linear combination of constantly many matrix entries. We also show that the size of our constructed graph is worst-case optimal among all planar graphs.

1 Introduction

Monge property, named after the 18th century mathematician Gaspard Monge, roughly say that the sum of shortest-path distances between two crossing pairs of points (x, y) and (z, w) is at least the sum of the ones between corresponding non-crossing pairs (x, z) and (y, w). The original motivation is to study the optimal transport of masses in the plane [31,40]. As a simple consequence of the Jordan curve theorem, Monge property has been tremendously helpful in designing efficient algorithms for *planar* optimization problems—whether the input is a planar graph or geometric objects lying in the plane [12, 25, 26, 39, 43]. Most famously, Monge property is central to the design of the *SMAWK algorithm* [2] for row-minimum queries in totally monotone matrices and the *Monge heap* data structure [28] for speeding up various optimization algorithms on planar and surface graphs [15, 28, 30, 33, 35, 41]. In some problems where Monge property is evident, it is not clear whether the problem has an obvious connection to planar metrics. Examples are fast dynamic programming using quadrangle inequalities [6, 29], as well as string problems such as the edit distance and longest common subsequence [46, 50]. (See Burkard *et al.* [11, 12], Park [43], and the citations within for additional applications of the Monge properties.) A characterization of matrices satisfying the Monge property is known to exist [7, 10, 45], but the following fundamental question relating planar metric to Monge property remains unanswered: *Given a metric between a finite number of points satisfying some Monge property, is the metric planar?*

We answer this question affirmatively. We show that given any distance matrix satisfying the (cyclic) Monge property, one can construct an edge-weighted planar graph realizing entries of the matrix *exactly* as graph distances between some subset of vertices (called *terminals*). In other words, we construct a *planar emulator* for any (cyclic) Monge matrix with zero diagonals. Moreover, the construction is optimal in size and takes time linear in the size of the distance matrix. In fact, each edge in the graph along with its weight is determined by a constant number of entries in the matrix. Such property is of independent interest and might be useful in designing efficient algorithms under various computation models.

1.1 Related work

Sketching graph distances. Emulators—arbitrary graphs that preserve distances between terminals in the input graph—are known to exist in general [8, 9, 19]. But without additional assumptions on the input graph there is a linear lower-bound on the size of the emulator (with respect to the size of the input graph) when the number of terminals is a polynomial $\Theta(n^\alpha)$ for some range of α strictly less than 1 [19].[1] Chang, Gawrychowski, Mozes, and Weimann [14] constructed the first sub-linear size emulator for any undirected unweighted planar graph: given any k-terminal planar graph with n vertices, an emulator of size $\tilde{O}(\min\{k^2, (kn)^{1/2}\})$ can be constructed in $\tilde{O}(n)$ time, which is optimal up to logarithmic factors.

A related structure, called a *spanner*, which preserves the distances approximately up to additive or multiplicative errors, is relatively well-understood for general graphs [9, 32, 44, 49, 51]. Spanners with stronger guarantees exist for geometrically/topologically constrained graphs [4, 13, 24, 38]. Similarly, *distance oracles* that answer distance queries exactly or approximately are known to exist for planar and surface graphs [1,5,16,28,36,37,42,47,48]. (See Ahmed *et al.* [3] for a recent survey on distance sketching.)

Circular planar graphs. One of the central problems in the theory of circular planar graphs considers the following problem: Given measures of effective resistances between all pairs of terminals, can we reconstruct

*Duke University, USA.
†Michigan State University, USA.

[1] Interestingly, when the number of terminals is barely sublinear (say $n/2^{\Theta(\log^* n)}$) in an undirected unweighted graph, there is a strictly sublinear-size emulator [8].

a planar resistor network realizing the measures where the terminals lie on the boundary? Colin de Verdière *et al.* [17, 18] and Curtis *et al.* [21, 22] showed that the reconstruction problem can be solved precisely when the effective resistance matrix is *totally non-negative*. The problem sounds similar to ours in spirit; in fact, when looking closer, the planar emulator problem is equivalent to their reconstruction problem in the $(\min, +)$-semiring instead of the standard $(+, \times)$-ring. The techniques involved in proving their theorem rely crucially on the fact that the weights are over a $(+, \times)$-ring and therefore do not apply to our problem.

1.2 Preliminaries

Monge properties. A matrix M satisfies the *Monge property* if for any two rows $i < i'$ and two columns $j < j'$, one has

$$M[i, j] + M[i', j'] \leq M[i', j] + M[i, j'].$$

Matrix M satisfies the *anti-Monge property* if the sign of the above inequality flipped. We often reorder the terms in the inequality to emphasize the monotonicity on the entry differences:

$$M[i', j'] - M[i, j'] \leq M[i', j] - M[i, j].$$

For the purpose of this paper we only consider *distance matrices*, where the diagonal entries are all zeros, the entries are symmetric and satisfy the triangle inequality. A distance matrix M is *cyclic Monge*[2] if for any four indices i, i', j, j' in cyclic order (that is, $i \leq i' \leq j \leq j'$ after some cyclic reordering of $[i, i', j, j']$), one has

$$M[i, j'] + M[i', j] \leq M[i, j] + M[i', j'].$$

(Notice the inequality sign flipped comparing to the standard Monge property.) Let M be a cyclic Monge distance matrix and let A and B be two disjoint subintervals of the index set of M. Then the submatrix of M between A and B must be an (anti-)Monge matrix.

Planar emulators. Consider an undirected planar graph G with edge weights and let ∂G be the vertices on the boundary of the outer face of G. We consider the distance matrix M between vertices in ∂G: for any pair of vertices i and j in ∂G, we set $M[i, j]$ to be the distance between i and j in G.

It is not immediately clear that any cyclic Monge distance matrix M comes as a distance matrix generated from some planar graph G. A *planar emulator* for a distance matrix M is a graph G whose vertex set $V(G)$ contains the indices of M (and possibly others), and the graph distance $d_G(u, v)$ between any pair of vertices u and v in G is equal to $M[u, v]$. Planarity and the Jordan curve theorem ensures that any distance matrix M of a planar emulator must satisfy the cyclic Monge property. Our main result shows that the converse is also true: *any cyclic Monge distance matrix admits a planar emulator*.

In Section 2 we describe the construction and prove its correctness. We show that the size of the construction is optimal in Section 3, and conclude the paper in Section 4.

[2] This is known as the *Kalmanson matrix* [23, 34], which is slightly more restricted than a *triangular Monge matrix* [12] or the *convex quadrangle inequality* [27].

2 Constructing a planar emulator

The goal of this section is to construct planar emulators for arbitrary cyclic Monge distance matrices.

Theorem 1 *Given any $n \times n$ cyclic Monge distance matrix M, there is a planar emulator for M with $\binom{n}{2}$ edges.*

For any given positive integer n, we define a planar graph G^n as follows (see Figure 1). Let the vertices of G^n be the set $\{v_{i,j}\}$, where i ranges in $[1 : n]$ and j ranges in $[1 : \min\{i, n - i + 1\}]$. Define *terminal* p_i to be $v_{i,\min\{i,n-i+1\}}$. The edges of G^n consist of *horizontal* edges and *vertical* edges. A *horizontal edge* $e^{\leftrightarrow}_{i,j}$ lies between each $v_{i,j}$ and $v_{i+1,j}$ where j ranges in $[1 : \lfloor n/2 \rfloor]$ and i ranges in $[j : n - j]$. A *vertical edge* $e^{\updownarrow}_{i,j}$ lies between each $v_{i,j}$ and $v_{i,j+1}$ where j ranges in $[1 : \min\{i, n + 1 - i\} - 1]$ and i ranges in $[2 : n - 1]$.

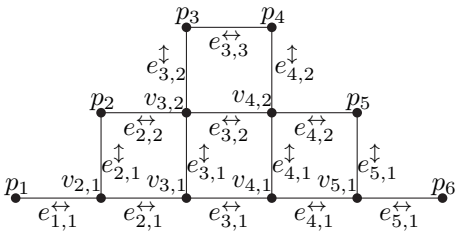


Figure 1: Graph G^6.

Consider a cyclic Monge distance matrix M and for brevity denote $M_{i,j} := M[i, j]$. We define the graph G^n_M as an edge-weighted copy of G^n, where the weight of a horizontal edge $e^{\leftrightarrow}_{i,j}$ is

$$\omega(e^{\leftrightarrow}_{i,j}) := \frac{1}{2}\left(M_{i+1,j} - M_{i,j} + M_{i,n-j+1} - M_{i+1,n-j+1}\right),$$

and the weight of a vertical edge $e^{\updownarrow}_{i,j}$ is

$$\omega(e^{\updownarrow}_{i,j}) := \frac{1}{2}\big(M_{i,j} - M_{i,j+1} + M_{i,n-j+1} - M_{i,n-j} + M_{j+1,n-j} - M_{j,n-j+1}\big).$$

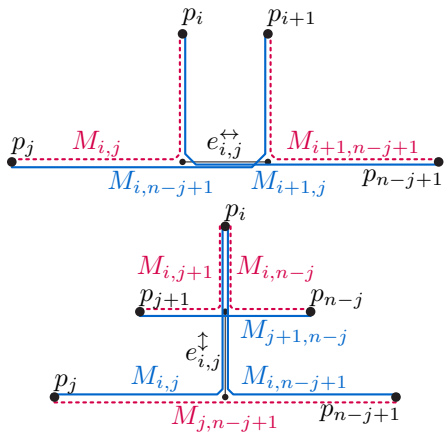


Figure 2: Values used to assign weights to $e^{\leftrightarrow}_{i,j}$ and $e^{\updownarrow}_{i,j}$.

(See Figure 2.) Henceforth, we will refer to the edge-weighted graph G^n_M as the *canonical realization* of M.

For the rest of the section, we show that $G \coloneqq G^n_M$ is a planar emulator of M. For this, it suffices to show that $d_G(p_i, p_j) = M[i, j]$ for all pairs of terminals p_i and p_j. First, we derive some properties of G using the fact that M is a cyclic Monge matrix.

Lemma 2 *If M is a cyclic Monge matrix, then all edge weights of G^n_M are non-negative.*

Proof. An edge of G^n_M is either horizontal or vertical. For any horizontal edge $e^{\leftrightarrow}_{i,j}$, the cyclic Monge property states that $M_{i,j} + M_{i+1,n-j+1} \le M_{i+1,j} + M_{i,n-j+1}$, and therefore $2\omega(e^{\leftrightarrow}_{i,j}) = M_{i+1,j} - M_{i,j} + M_{i,n-j+1} - M_{i+1,n-j+1} \ge 0$.

For any vertical edge $e^{\updownarrow}_{i,j}$, the cyclic Monge property states that (1) $M_{i,j+1} + M_{j,n-j} \le M_{i,j} + M_{j+1,n-j}$ and (2) $M_{i,n-j} + M_{j,n-j+1} \le M_{j,n-j} + M_{i,n-j+1}$. Combining (1) and (2) gives $2\omega(e^{\updownarrow}_{i,j}) = M_{i,j} - M_{i,j+1} + M_{i,n-j+1} - M_{i,n-j} + M_{j+1,n-j} - M_{j,n-j+1} \ge 0$. □

It follows that the minimum-weight path from p_i to p_j in G is simple.

Next, we show that there is at least one path from p_i to p_j achieving the cost $M[i, j]$. For $i \le i'$, the path of horizontal edges between $v_{i,j}$ and $v_{i',j}$ in G has weight

$$\sum_{x\in[i:i'-1]} \omega(e^{\leftrightarrow}_{x,j}) = \frac{1}{2} \sum_{x\in[i:i'-1]} \big(M_{x+1,j} - M_{x,j} + M_{x,n-j+1} - M_{x+1,n-j+1}\big) = \frac{1}{2}\left(M_{i',j} - M_{i,j} + M_{i,n-j+1} - M_{i',n-j+1}\right),$$

and for $j \le j'$, the path of vertical edges between $v_{i,j}$ and $v_{i,j'}$ has weight

$$\sum_{y\in[j:j'-1]} \omega(e^{\updownarrow}_{i,y}) = \frac{1}{2} \sum_{y\in[j:j'-1]} \big(M_{i,y} - M_{i,y+1} + M_{i,n-y+1} - M_{i,n-y} + M_{y+1,n-y} - M_{y,n-y+1}\big) = \frac{1}{2}\big(M_{i,j} - M_{i,j'} + M_{i,n-j+1} - M_{i,n-j'+1} + M_{j',n-j'+1} - M_{j,n-j+1}\big).$$

Consider two terminals p_i and p_j and assume that $\min\{i, n-i+1\} \ge \min\{j, n-j+1\}$. Let $\pi_{j,i}$ be the unique L-shaped (simple) path from p_j to p_i that consists of a path $\pi^{\leftrightarrow}_{j,i}$ of horizontal edges followed by a path $\pi^{\updownarrow}_{j,i}$ of vertical edges (both paths might possibly be empty). When $\min\{i, n-i+1\} > \min\{j, n-j+1\}$ we define $\pi_{j,i} \coloneqq \pi_{i,j}$.

Lemma 3 *Let M be a cyclic Monge distance matrix. The weight of $\pi_{j,i}$ in G^n_M is $M_{i,j}$.*

Proof. We assume that $j \le \lceil n/2 \rceil$ (the other case is symmetric). The vertex at the end of $\pi^{\leftrightarrow}_{j,i}$ (and at the start of $\pi^{\updownarrow}_{j,i}$) is $v_{i,j}$. Let $i' \coloneqq \min\{i, n-i+1\}$, then the weight of $\pi_{j,i}$ is

$$\begin{aligned}\omega(\pi_{j,i}) &= \sum_{x\in[j:i-1]} \omega(e^{\leftrightarrow}_{x,j}) + \sum_{y\in[j:i'-1]} \omega(e^{\updownarrow}_{i,y}) \\ &= \frac{1}{2}\big((M_{i,j} - M_{j,j} + M_{j,n-j+1} - M_{i,n-j+1}) + \\ &\qquad (M_{i,j} - M_{i,i'} + M_{i,n-j+1} - M_{i,n-i'+1} + \\ &\qquad M_{i',n-i'+1} - M_{j,n-j+1})\big) \\ &= \frac{1}{2}(M_{i,j} + M_{i,j} - M_{i,i'} - M_{i,n-i'+1} + M_{i',n-i'+1}),\end{aligned}$$

where either $M_{i,i'} = 0$ and $M_{i,n-i'+1} = M_{i',n-i'+1}$, or $M_{i,n-i'+1} = 0$ and $M_{i,i'} = M_{i',n-i'+1}$; so $\omega(\pi_{j,i}) = M_{i,j}$. □

By Lemma 3 we have $d_G(p_i, p_j) \le M_{i,j}$, so it remains to show that $d_G(p_i, p_j) \ge M_{i,j}$. Define the y-coordinate of a horizontal edge $e^{\leftrightarrow}_{i,j}$ as j, and the x-coordinate of a vertical edge $e^{\updownarrow}_{i,j}$ as i. We next show that G contains a minimum-weight path from p_i to p_j whose horizontal edges all have the same y-coordinate. It follows that there is a minimum-weight path consisting of at most one subpath of horizontal edges.

Lemma 4 *Let M be a cyclic Monge distance matrix. For any pair of terminals p and p', G^n_M has a minimum-weight path from p to p' whose horizontal edges all have the same y-coordinate.*

Proof. For a path π, let $\sigma(\pi)$ be the sum of y-coordinates of its horizontal edges. Let α be a minimum-weight path from p to p' that minimizes $\sigma(\alpha)$ (over all minimum-weight paths from p to p'). We

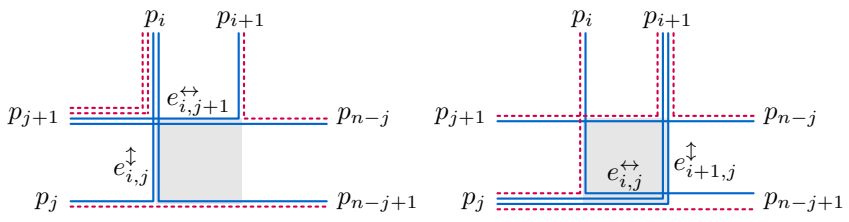


Figure 3: The sum of weights of $e^{\leftrightarrow}_{i,j}$ and $e^{\updownarrow}_{i+1,j}$ is at most that of $e^{\updownarrow}_{i,j}$ and $e^{\leftrightarrow}_{i,j+1}$.

Figure 4: The weight of the horizontal path from $v_{i,j+1}$ to $v_{i',j+1}$ is at most the total weight of $e^{\updownarrow}_{i,j}$, $e^{\updownarrow}_{i',j}$, and the horizontal path from $v_{i,j}$ to $v_{i',j}$.

claim that all horizontal edges of α have the same y-coordinate. Suppose not, then α contains a two-edge subpath consisting of a vertical edge $e^{\updownarrow}_{i,j}$ and a horizontal edge $e^{\leftrightarrow}_{i,j+1}$ or $e^{\leftrightarrow}_{i-1,j+1}$. We consider only the case where the subpath has edges $e^{\updownarrow}_{i,j}$ and $e^{\leftrightarrow}_{i,j+1}$ (the other case is symmetric). Consider the path β obtained from α by replacing this subpath by $e^{\leftrightarrow}_{i,j}$ and $e^{\updownarrow}_{i+1,j}$. Then $\sigma(\beta) < \sigma(\alpha)$, so by assumption β cannot be a minimum-weight path. However, Figure 3 shows that the weight of β is at most that of α, contradicting that α is a minimum-weight path that minimizes σ. □

Finally, we show that there is a minimum-weight path for which additionally, its vertical edges all have the same x-coordinate. Together with the fact that all edge weights are non-negative (Lemma 2), it follows that $\pi_{j,i}$ is a minimum-weight path between p_j and p_i.

Lemma 5 *Let M be a cyclic Monge distance matrix. For any pair of terminals p and p', G^n_M has a minimum-weight path from p to p' whose horizontal edges all have the same y-coordinate, and whose vertical edges all have the same x-coordinate.*

Proof. By Lemma 4, there is a minimum-weight path from p to p' whose horizontal edges all have the same y-coordinate, and without loss of generality assume that this y-coordinate is maximal over all such paths. Because all edges have nonnegative weights by Lemma 2, we may assume that this path consists of a path of vertical edges (with decreasing y-coordinates), followed by a path of horizontal edges whose x-coordinates are increasing or decreasing, and finally a path of vertical edges with increasing y-coordinates. Suppose that the subpath of horizontal edges is surrounded by vertical edges $e^{\updownarrow}_{i,j}$ and $e^{\updownarrow}_{i',j}$ with $i < i'$ (the case $i > i'$ is symmetric). Let α be the path consisting of $e^{\updownarrow}_{i,j}$, the edges $e^{\leftrightarrow}_{x,j}$ for $i \leq x < i'$, and $e^{\updownarrow}_{i',j}$; let β be the path of edges $e^{\leftrightarrow}_{x,j+1}$ for $i \leq x < i'$. Apply cyclic Monge property twice, one can show that $2M_{i',j} + 2M_{j+1,n-j} - M_{i',j+1} + 2M_{i,n-j+1} - 2M_{j,n-j+1} - M_{i,n-j} \geq M_{i',j+1} + M_{i,n-j}$, which implies that the weight of β is at most that of α, so replacing α by β yields a shortest path whose horizontal edges all have the same y-coordinate, but one bigger than that of the horizontal edges of α, which is a contradiction. (See Figure 4.) □

As an immediate corollary of Lemmas 2, 3, and 5, every $n \times n$ cyclic Monge distance matrix has a planar emulator of size $\binom{n}{2}$, proving Theorem 1.

3 Lower bound on the size of planar emulators

In this section we show that some Monge distance matrices requires $\binom{n}{2}$ edges in any of its planar emulator. A similar result by Cossarini [20] says that any planar emulator of some *cyclic* Monge matrix requires $\binom{n}{2}$ edges. Therefore, our canonical realization is worst-case optimal in size.

Theorem 6 *Some $n \times n$ Monge distance matrices have no planar emulator with fewer than $\binom{n}{2}$ edges.*

Proof. Let M be a Monge distance matrix. The vector $(M_{i,j})_{i<j} \in \mathbb{R}^{\binom{n}{2}}$ completely determines M since

$M_{i,i} = 0$ and $M_{i,j} = M_{j,i}$ as d is a graph metric on the canonical realization of M. The set of such vectors over all Monge distance matrices yields a convex polytope $\mathcal{P}$, as it is bounded only by the hyperplanes arising from the linear inequalities of the triangle inequality and cyclic Monge property. We show that $\mathcal{P}$ is $\binom{n}{2}$-dimensional.

For this, we define a family of $\binom{n}{2}$ sets $(E_e)_{e\in E(G)}$ of edges indexed by the edges of G^n_M. For each horizontal edge $e^{\leftrightarrow}_{i,j}$, let $E_{e^{\leftrightarrow}_{i,j}} \coloneqq \{e^{\leftrightarrow}_{i,j'} \mid j' \le j\}$. For each vertical edge $e^{\updownarrow}_{i,j}$, let $E_{e^{\updownarrow}_{i,j}} \coloneqq \{e^{\updownarrow}_{i,j}\} \cup E_{e^{\leftrightarrow}_{i,j}} \cup E_{e^{\leftrightarrow}_{i+1,j}}$. For each edge e, define the weight function ω_e as the characteristic function of E_e; in other words, let $\omega_e : E \to \{0,1\}$, with $\omega_e(e') = 1$ if $e' \in E_e$, and $\omega_e(e') = 0$ otherwise. We show that the $\binom{n}{2}$ weight functions $(\omega_e)_{e\in E(G)}$ are linearly independent. For each horizontal edge $e^{\leftrightarrow}_{i,1}$, $\omega_{e^{\leftrightarrow}_{i,1}}$ sets only the weight of edge $e^{\leftrightarrow}_{i,1}$ to one, and all other edges to zero. Similarly, for each horizontal edge $e^{\leftrightarrow}_{i,j}$ with $j > 1$, $e \mapsto \omega_{e^{\leftrightarrow}_{i,j}}(e) - \omega_{e^{\leftrightarrow}_{i,j-1}}(e)$ sets only the weight of edge $e^{\leftrightarrow}_{i,j}$ to one. Finally, for each vertical edge $e^{\updownarrow}_{i,j}$, $e \mapsto \omega_{e^{\updownarrow}_{i,j}}(e) - \omega_{e^{\leftrightarrow}_{i,j}}(e) - \omega_{e^{\leftrightarrow}_{i+1,j}}(e)$ sets only the weight of edge $e^{\updownarrow}_{i,j}$ to one. Since each of the $\binom{n}{2}$ edges can be set to weight one while all other edges are set to zero, the defined weight functions are linearly independent, and moreover, any weight function can be obtained as a linear combination of $(\omega_e)_{e\in E(G)}$.

Since the polytope $\mathcal{P}$ is $\binom{n}{2}$-dimensional, there exists a Monge distance matrix whose entries are in general position: there is no indexed family S of fewer than $\binom{n}{2}$ real numbers such that each of the $\binom{n}{2}$ distances can be written as the sum of a subset of S. Since the length of each shortest path in a nonnegatively edge-weighted graph is the sum of a subset of its edge-weights, there is a Monge distance matrix that does not have a planar emulator with fewer than $\binom{n}{2}$ edges. □

The argument of Theorem 6 relies on the fact that the set of distances can be chosen to lie in general position. We present a different, but slightly weaker lower bound for the more general setting where the weights are integers up to $\lceil n/2 \rceil$. A Monge matrix M is *unit-Monge* if for all i and j,

$$M[i+1,j] - M[i,j] \in \{-1,0,1\}, \text{ and}$$
$$M[i,j] - M[i,j+1] \in \{-1,0,1\}.$$

Theorem 7 *Some $n \times n$ unit-Monge distance matrices have no planar emulator with fewer than $n^2/8 + n/2$ edges.*

Proof. Let M be a distance matrix defined as follows. Consider a rectangular grid graph with vertex set $\{0,\dots,w\} \times \{0,\dots,h\}$ and edges between vertices at distance 1, so that vertex (x,y) has (unit-weight) edges to $(x \pm 1, y)$ and $(x, y \pm 1)$. For all y and k, we have $d((0,y),(w,y \pm k)) = w + k$, and symmetrically $d((x,0),(x \pm k,h)) = h + k$ for all x and k. Let M be the distance matrix from the set of vertices $\{(x,0)\} \cup \{(0,y)\}$ to the set of vertices $\{(x,h)\} \cup \{(w,y)\}$; distance matrix M must be unit-Monge.

Consider an arbitrary planar emulator G of M. Let d_G denote the shortest-path metric on G. For vertices i,j,k,ℓ in clockwise-order along the outer face, we have $d_G(i,\ell) + d_G(j,k) \le d_G(i,k) + d_G(j,\ell)$. On the other hand, for any pair of points p and q where p is on a shortest path from i to ℓ and q on a shortest path from j to k, we have $d_G(i,\ell) + d_G(j,k) + 2d_G(p,q) \ge d_G(i,k) + d_G(j,\ell)$.

Denote by $\pi^{\leftrightarrow}_y$ a shortest path in G between $(0,y)$ and (w,y), and by $\pi^{\updownarrow}_x$ a shortest path in G between $(x,0)$ and (x,h). We will show that the paths $\pi^{\updownarrow}_x$ are disjoint and have h edges each. Recall that $d_G(i,\ell) + d_G(j,k) + 2d_G(p,q) \ge d_G(i,k) + d_G(j,\ell)$, so

$$\begin{aligned}
&\|\pi^{\leftrightarrow}_y\| + \|\pi^{\leftrightarrow}_{y+k}\| + 2d_G(\pi^{\leftrightarrow}_y, \pi^{\leftrightarrow}_{y+k}) \\
&= 2w + 2d_G(\pi^{\leftrightarrow}_y, \pi^{\leftrightarrow}_{y+k}) \\
&\ge d_G((0,y),(w,y+k)) + d_G((0,y+k),(w,y)) \\
&= 2(w+k),
\end{aligned}$$

and thus any pair of points $p \in \pi^{\leftrightarrow}_y$ and $q \in \pi^{\leftrightarrow}_{y+k}$ on distinct paths have distance at least $k \ge 1$, so different such paths are vertex-disjoint. Any path $\pi^{\updownarrow}_x$ must cross all the (vertex-disjoint) paths $\pi^{\leftrightarrow}_0, \dots \pi^{\leftrightarrow}_h$, and thus have at least h edges (not shared with any path $\pi^{\leftrightarrow}_y$) of length at least 1. Therefore, the paths $\pi^{\updownarrow}_x$ and $\pi^{\leftrightarrow}_y$ (over all x and by symmetric argument y) contain at least $(w+1)h + (h+1)w$ edges. We have $n = 2(w+h)$; by taking $w = h = n/4$, this yields a lower bound of

$$2(n/4+1)(n/4) = n^2/8 + n/2$$

edges for any planar emulator of M. □

We remark that the argument of Theorem 7 depends only on distances between opposite sides of the grid, and can be made to depend only on the linearly many distances $d((0,y),(w,y+k))$ and $d((x,0),(x+k,h))$ with $k \in \{-1,0,1\}$.

Cossarini [20] proved that any planar emulator for some $n \times n$ *cyclic* unit-Monge matrix must have at least $\binom{n}{2}$ edges. Our result, while slightly weaker in comparison, applies to general unit-Monge matrices, which can be viewed as the *directed* version of the problem.

4 Discussion

In this paper we have shown that any cyclic Monge distance matrix admits a quadratic-size planar emulator. Our construction is universal in the sense that the underlying graph does not depend on the entries of the matrix. And there are metrics for which each edge must

be used by some shortest path. We also showed that already for planar emulators of unit-Monge distance matrices (which can be represented in linear space), $\Omega(n^2)$ edges are sometimes necessary.

The cyclic-Monge distance matrices considered in this paper are closely connected to the set of intrinsic metrics of topological disks. In particular, a given metric on points in a circle can be realized as a metric intrinsic to a topological disk bounded by that circle if and only if the metric is a cyclic-Monge distance matrix. We conclude with an open problem.

- Under what conditions do surfaces other than the disk (such as the Möbius strip, or a torus with holes) realize a given metric between points on their boundary? Do such surfaces also have a universal emulator, and if so, one with at most $\binom{n}{2}$ edges?

References

[1] A. Abboud, P. Gawrychowski, S. Mozes, and O. Weimann. Near-optimal compression for the planar graph metric. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 530–549, 2018.

[2] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, Nov. 1987.

[3] R. Ahmed, G. Bodwin, F. D. Sahneh, K. Hamm, M. J. L. Jebelli, S. Kobourov, and R. Spence. Graph spanners: A tutorial review. Sept. 2019.

[4] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, Jan. 1993.

[5] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Algorithms — ESA '96*, pages 514–528, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[6] W. Bein, M. J. Golin, L. L. Larmore, and Y. Zhang. The Knuth-Yao quadrangle-inequality speedup is a consequence of total monotonicity. *ACM Transactions on Algorithms*, 6(1):1–22, Dec. 2009.

[7] W. W. Bein and P. K. Pathak. A characterization of the Monge property and its connection to statistics. *Demonstratio Mathematica*, 29(2):451–457, Apr. 1996.

[8] G. Bodwin. Linear Size Distance Preservers. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 600–615. Society for Industrial and Applied Mathematics, Jan. 2017.

[9] G. Bodwin and V. V. Williams. Better distance preservers and additive spanners. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 855–872. Society for Industrial and Applied Mathematics, Jan. 2016.

[10] V. Y. Burdyuk and V. Trofimov. Generalization of results of Gilmore and Gomory on solution of traveling salesman problem. *Engineering Cybernetics*, 14(3):12–18, 1976.

[11] R. E. Burkard. Monge properties, discrete convexity and applications. *European Journal of Operational Research*, 176(1):1–14, Jan. 2007.

[12] R. E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, Sept. 1996.

[13] N. Catusse, V. Chepoi, and Y. Vaxès. Planar hop spanners for unit disk graphs. In C. Scheideler, editor, *Algorithms for Sensor Systems*, volume 6451, pages 16–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[14] H.-C. Chang, P. Gawrychowski, S. Mozes, and O. Weimann. Near-optimal distance emulator for planar graphs. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[15] J. Łacki and P. Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In *Algorithms – ESA 2011*, volume 6942, pages 155–166. Springer Berlin Heidelberg, 2011.

[16] V. Cohen-Addad, S. Dahlgaard, and C. Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 962–973, Berkeley, CA, Oct. 2017.

[17] Y. Colin de Verdière, I. Gitler, and D. Vertigan. Réseaux électriques planaires II. *Commentarii Mathematici Helvetici*, 71(1):144–167, Dec. 1996.

[18] Y. D. Colin de Verdière. Réseaux électriques planaires I. *Commentarii Mathematici Helvetici*, 69:351–374, Dec. 1994.

[19] D. Coppersmith and M. Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM Journal on Discrete Mathematics*, 20(2):463–501, 2006.

[20] M. Cossarini. *Discrete Surfaces with Length and Area and Minimal Fillings of the Circle*. Ph.D. Dissertation, Instituto Nacional de Matematica Pura e Aplicada, Sept. 2018.

[21] E. Curtis, E. Mooers, and J. Morrow. Finding the conductors in circular networks from boundary measurements. *ESAIM: Mathematical Modelling and Numerical Analysis*, 28(7):781–814, 1994.

[22] E. B. Curtis, D. Ingerman, and J. A. Morrow. Circular planar graphs and resistor networks. *Linear Algebra and its Applications*, 283(1):115–150, Nov. 1998.

[23] V. G. Deĭneko, J. A. Van der Veen, R. Rudolf, and G. J. Woeginger. Three easy special cases of the euclidean travelling salesman problem. *RAIRO - Operations Research*, 31(4):343–362, 1997.

[24] F. F. Dragan, F. V. Fomin, and P. A. Golovach. Spanners in sparse graphs. *Journal of Computer and System Sciences*, 77(6):1108–1119, Nov. 2011.

[25] T. Dudás and R. Rudolf. Spanning trees and shortest paths in Monge graphs. *Computing*, 60(2):109–119, June 1998.

[26] D. Eppstein. Sequence comparison with mixed convex and concave costs. *Journal of Algorithms*, 11(1):85–101, Mar. 1990.

[27] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic programming II: Convex and concave cost functions. *Journal of the ACM*, 39(3):546–567, July 1992.

[28] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, Aug. 2006.

[29] Z. Galil and K. Park. Dynamic programming with convexity, concavity and sparsity. *Theoretical Computer Science*, 92(1):49–76, Jan. 1992.

[30] P. Gawrychowski, S. Mozes, and O. Weimann. Submatrix maximum queries in Monge and partial Monge matrices are equivalent to predecessor search. *ACM Transactions on Algorithms*, 16(2):16:1–16:24, Apr. 2020.

[31] A. J. Hoffman. On simple linear programming problems. In *Proceedings of Symposia in Pure Mathematics*, volume 7, pages 317–327. AMS, 1963.

[32] S.-E. Huang and S. Pettie. Lower Bounds on Sparse Spanners, Emulators, and Diameter-reducing shortcuts. In *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, pages 26:1–26:12, 2018.

[33] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC '11)*, pages 313–322, San Jose, California, USA, 2011.

[34] K. Kalmanson. Edgeconvex Circuits and the Traveling Salesman Problem. *Canadian Journal of Mathematics*, 27(5):1000–1010, Oct. 1975.

[35] H. Kaplan, S. Mozes, Y. Nussbaum, and M. Sharir. Submatrix maximum queries in Monge matrices and partial Monge matrices, and their applications. *ACM Transactions on Algorithms*, 13(2):26:1–26:42, Mar. 2017.

[36] K.-i. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming*, pages 135–146, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[37] P. N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 146–155, 2005.

[38] P. N. Klein. A subset spanner for planar graphs, with application to subset TSP. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 749–756, New York, NY, USA, 2006. ACM.

[39] L. L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. *Journal of Algorithms*, 12(3):490–515, Sept. 1991.

[40] G. Monge. *Mémoire sur la Théorie des Déblais et des Remblais*. De l'Imprimerie Royale, 1781.

[41] S. Mozes, C. Nikolaev, Y. Nussbaum, and O. Weimann. Minimum cut of directed planar graphs in $O(n \log \log n)$ time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 477–494, New Orleans, Louisiana, Jan. 2018.

[42] S. Mozes and C. Sommer. Exact Distance Oracles for Planar Graphs. In Y. Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 209–222, Philadelphia, PA, Jan. 2012. Society for Industrial and Applied Mathematics.

[43] J. K. Park. *The Monge Array: An Abstraction and Its Applications*. Ph.D. dissertation, MIT, 1991.

[44] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[45] R. Rudolf and G. J. Woeginger. The cone of Monge matrices: Extremal rays and applications. *ZOR – Methods and Models of Operations Research*, 42(2):161–168, June 1995.

[46] L. M. Russo. Monge properties of sequence alignment. *Theoretical Computer Science*, 423:30–49, Mar. 2012.

[47] C. Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4):1–31, Mar. 2014.

[48] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, Nov. 2004.

[49] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 802–809, Miami, Florida, 2006. ACM Press.

[50] A. Tiskin. Semi-local string comparison: Algorithmic techniques and applications. Nov. 2013.

[51] D. Woodruff. Lower bounds for additive spanners, emulators, and more. In *Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 389–398, 2006.

Simultaneous Visibility Representations of Undirected Pairs of Graphs*

Ben Chugg† William S. Evans‡ Kelvin Wong§

Abstract

We consider the problem of determining if a pair of undirected graphs $\langle G_\mathsf{v}, G_\mathsf{h}\rangle$, which share the same vertex set, has a representation using opaque geometric shapes for vertices, and vertical/horizontal visibility between shapes for edges. While such a simultaneous visibility representation of two graphs can be determined efficiently if the direction of the required visibility for each edge is provided (and the vertex shapes are sufficiently simple), it was unclear if edge direction is critical for efficiency. We show that the problem is NP-complete without that information, even for graphs that are only slightly more complex than paths. In addition, we characterize which pairs of paths have simultaneous visibility representations using fixed orientation L-shapes. This narrows the range of possible graph families for which determining simultaneous visibility representation is non-trivial yet not NP-hard.

1 Introduction

A *visibility representation* Γ of a graph $G = (V, E)$ is a set of disjoint geometric objects $\{\Gamma(v) \mid v \in V\}$ representing vertices chosen from a family of allowed objects (e.g., axis-aligned rectangles in the plane) where $\Gamma(u)$ sees $\Gamma(v)$ if and only if $uv \in E$. Typically, the meaning of "sees" is that there exists a line segment (perhaps axis-aligned, perhaps positive width) from $\Gamma(u)$ to $\Gamma(v)$ that does not intersect $\Gamma(w)$ for any other $w \in V$; such a line segment is called a *line-of-sight*. Many different classes of visibility representations may be defined by changing the family of allowed objects and the meaning of "sees." For example, *bar visibility representations* (BVRs) use horizontal line segments as vertices and vertical lines-of-sight for edges [9, 11, 21, 20, 7, 14, 15]; *rectangle visibility representations* (RVRs) use (solid) axis-aligned rectangles and axis-aligned lines-of-sight [8, 22, 18, 2, 5, 16]; and *unit square visibility representations* (USVRs) use axis-aligned unit squares and axis-aligned lines-of-sight [4]. The popularity of this type of graph representation lies in its potential applicability to problems in VLSI design and the production of readable representations of planar and non-planar graphs. Determining which graphs or families of graphs have visibility representations of a particular type is a fascinating area of research.

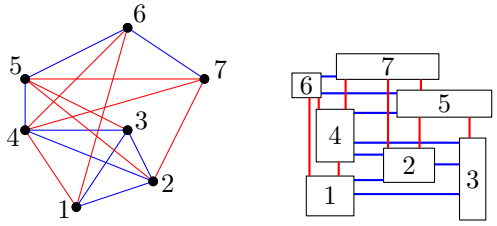


Figure 1: Left: A pair of graphs $\langle G_\mathsf{v}, G_\mathsf{h}\rangle$ on the same vertex set. Red (light) edges are those of G_v, while blue (darker) are those of G_h. Right: A simultaneous visibility representation of $\langle G_\mathsf{v}, G_\mathsf{h}\rangle$ using rectangles.

Our focus in this work is on the *simultaneous* visibility representation of pairs of graphs that share the same vertex set (see Fig. 1). A *simultaneous visibility representation* (SVR) of $G_\mathsf{v} = (V, E_\mathsf{v})$ and $G_\mathsf{h} = (V, E_\mathsf{h})$ is a visibility representation Γ that, using vertical lines-of-sight, represents G_v and, using horizontal lines-of-sight, represents G_h. Streinu and Whitesides [19] describe a beautiful connection between a pair of *directed* planar graphs $\langle \overrightarrow{G}_\mathsf{v}, \overrightarrow{G}_\mathsf{h}\rangle$ and their planar duals that determines if the pair has a directed simultaneous visibility representation using rectangles (a directed RSVR) Γ, where an edge directed from u to v in $\overrightarrow{G}_\mathsf{v}$ or $\overrightarrow{G}_\mathsf{h}$ is realized by a low-to-high or left-to-right, respectively, line-of-sight from $\Gamma(u)$ to $\Gamma(v)$. Evans et al. [13] extended this to the family of geometric objects called *L-shapes*, which are the union of two axis-aligned segments in the plane that share a common endpoint and come in four orientations: $\{\llcorner, \lrcorner, \urcorner, \ulcorner\}$. They gave a polynomial time algorithm for determining if a pair of directed graphs $\langle \overrightarrow{G}_\mathsf{v}, \overrightarrow{G}_\mathsf{h}\rangle$ has a directed simultaneous visibility representation using L-shapes, Γ, in which the orientation of $\Gamma(v)$ is given by $\Phi : V \to \{\llcorner, \lrcorner, \urcorner, \ulcorner\}$ (a directed Φ-LSVR).

The complexity of determining if a pair of *undirected* graphs has a simultaneous visibility representation using L-shapes was stated as an open problem [13]. In this paper, we show (Section 3) that the problem is NP-complete. What is surprising about this result is the simplicity of the graphs for which the problem is hard: For L-shapes (and many other families of shapes includ-

*Supported by Canada NSERC Discovery Grant and Undergraduate Student Research Awards. Full paper available at https://arxiv.org/abs/2005.00937

†Stanford University, benchugg@stanford.edu
‡University of British Columbia, will@cs.ubc.ca
§University of Toronto and Uber ATG, kelvin.wong@uber.com

ing rectangles), one graph can be a set of disjoint paths and the other a set of disjoint copies of a tree with 3 leaves connected to a root by paths of length 2 (i.e. $\curlyvee$). For unit squares (and for translates of any specified connected shape with positive width and height), one graph can be a set of disjoint paths and the other a set of disjoint claws ($K_{1,3}$).

This limits the families of graphs for which we can reasonably hope to efficiently determine a simultaneous visibility representation. We describe a linear time algorithm (Section 4) that determines if a pair of undirected *paths* has a simultaneous visibility representation using L-shapes, all with the same orientation {⌞} (an LSVR). The algorithm is quite simple but relies on characterizing those pairs of paths for which such a representation is possible. The characterization of such pairs of paths for representations using rectangles, unit squares, and L-shapes with more than one orientation is easier.

Our work has aspects of both visibility representation and simultaneous geometric graph embedding (SGE). SGE is the problem of deciding, given a set of planar graphs on the same set of vertices, whether the vertices can be placed in the plane so that each graph has a straight-line drawing on the placed vertices. As in our problem, SGE, which is NP-hard [12], asks to represent several specified graphs using one common vertex set representation. However, the hardness result for SGE does not directly imply hardness of deciding simultaneous visibility representation. Similarly, deciding if a graph has an RVR [18] or a USVR [4] is NP-hard, but since the input does not specify which edges should be realized as vertical versus horizontal lines-of-sight, the problems are quite different. Choosing how the graph should be split into vertical and horizontal parts is an additional opportunity (or burden) for deciding if these representations exist.

Rather than requiring the visibility representation to partition the edges of the graph in a prescribed manner between vertical and horizontal visibilities, Biedl et al. [1] require that the visibility edges (lines-of-sight) obey the same embedding as a prescribed embedding of the original graph, which may include edge crossings. They can decide if such a restricted RVR exists in polynomial time, and in linear time if the graph is 1-planar. Di Giacomo et al. [10] show that deciding if a similarly restricted ortho-polygon[1] visibility representation exists for an embedded graph takes polynomial time as well.

2 Preliminaries

In this paper, we will assume that vertex shapes are connected and closed (rather than open) sets in the plane, and that lines-of-sight are 0-width (rather than positive-width) and exist between two shapes if and only if the corresponding vertices are connected by an edge. This implies that G_{v} and G_{h} must have *strong-visibility representations* [20] to have a simultaneous visibility representation. For visibility graphs, these choices make a difference since, for example, $K_{2,4}$ can be represented if lines-of-sight are positive-width (an *ϵ-visibility representation*) but does not have a strong-visibility representation [20]. However, for our results, we could adopt either model with only minor modifications to our proofs.

Let Γ be an SVR of $\langle G_{\mathsf{v}}, G_{\mathsf{h}}\rangle$. Given a subset $S \subset V$, we let $\Gamma(S) = \bigcup_{v\in S} \Gamma(v)$. For a vertex $v \in V$, let $X_\Gamma(v)$ and $Y_\Gamma(v)$ be the orthogonal projections of $\Gamma(v)$ onto the x-axis and y-axis respectively. For a set of vertices $S \subset V$, let $X_\Gamma(S) = \bigcup_{v\in S} X_\Gamma(v)$ and $Y_\Gamma(S) = \bigcup_{v\in S} Y_\Gamma(v)$. Set $\underline{x}_\Gamma(v) = \min X_\Gamma(v)$, $\overline{x}_\Gamma(v) = \max X_\Gamma(v)$, $\underline{y}_\Gamma(v) = \min Y_\Gamma(v)$, and $\overline{y}_\Gamma(v) = \max Y_\Gamma(v)$. We write $Y_\Gamma(u) \le Y_\Gamma(v)$ if $\overline{y}_\Gamma(u) \le \underline{y}_\Gamma(v)$ and $X_\Gamma(u) \le X_\Gamma(v)$ if $\overline{x}_\Gamma(u) \le \underline{x}_\Gamma(v)$. We also use the shorthand $[n]$ for $\{1, 2, \ldots, n\}$.

We state three basic properties of any visibility representation Γ of a graph $G = (V, E)$. For brevity, these properties are stated for vertical visibility representations only but also hold for horizontal visibility representations by symmetry. See Appendix A for proofs.

Property 1 *For $S_1, S_2 \subset V$ and $x \in X_\Gamma(S_1) \cap X_\Gamma(S_2)$, there exists a path $u = u_1, \ldots, u_k = v$ in G for some $u \in S_1$ and $v \in S_2$ such that $x \in X_\Gamma(u_i)$ for all $i \in [k]$.*

Property 2 *If an endpoint of $X_\Gamma(w)$ is strictly contained in $X_\Gamma(u) \cap X_\Gamma(v)$ and $\underline{y}_\Gamma(u) < \underline{y}_\Gamma(w) < \underline{y}_\Gamma(v)$ for $u, v, w \in V$, then there is a cycle in G.*

Property 3 *Let $u_1, \ldots, u_\ell$ be the only path from u_1 to u_ℓ in G. If $\Gamma(u_i)$ and $\Gamma(u_k)$ are both above or both below $\Gamma(u_j)$ for $i < j < k$, then $X_\Gamma(\{u_1, \ldots, u_i\}) \cap X_\Gamma(\{u_k, \ldots, u_\ell\}) = \emptyset$.*

3 Hardness

In this section, we study the complexity of determining if a pair of undirected graphs has an SVR. We first consider the problem of determining SVRs using unit squares (Section 3.1). Then, we discuss how our results can generalize to other connected shapes as well (Section 3.2 and Appendix C). In the case of L-shapes, our results settle an open question of Evans et al. [13].

To begin, we first state two lemmas that characterize how the gadgets in our hardness proofs can be drawn. See Appendix B for proofs and illustrations.

Lemma 1 *Let $G = (V, E)$ be a connected graph with a (vertical) visibility representation Γ. If $u \in V$ is a cut vertex whose removal creates components $C_1, \ldots, C_k$ then $X_\Gamma(C_i) \not\subseteq X_\Gamma(u)$ for at most two components.*

[1] a polygon whose edges are axis-aligned.

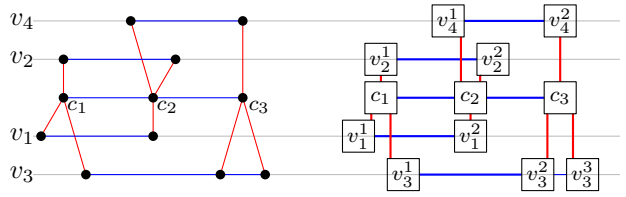


Figure 2: Left: G_{v} (red) and G_{h} (blue) for Monotone Not-All-Equal 3SAT instance $\varphi = (v_1 \vee v_2 \vee v_3)(v_4 \vee v_1 \vee v_2)(v_3 \vee v_4 \vee v_3)$. Right: A USSVR of $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$ encoding the truth assignment $v_2, v_4 = T$ and $v_1, v_3 = F$

Lemma 2 *Let $G = (V, E)$ be a graph with a (vertical) visibility representation Γ. If C_1 and C_2 are components in G, then either $X_\Gamma(C_1) > X_\Gamma(C_2)$[2], or vice versa.*

3.1 USSVR recognition

We first prove that determining if a pair of undirected graphs has a *simultaneous visibility representation using unit squares* (USSVR) is NP-complete.

Theorem 3 *Deciding if a pair of undirected graphs has a USSVR is NP-complete.*

For our proof, we reduce from the NP-complete problem of Monotone Not-All-Equal 3SAT [17]. This variant of 3SAT stipulates that every clause has three positive literals of which exactly one or two must be satisfied.

Construction. Let φ be an instance of Monotone Not-All-Equal 3SAT with a set $\mathcal{C}$ of m clauses and a set $\mathcal{V}$ of $n \leq 3m$ variables. All clauses form a path in G_{h} in the order of their appearance in φ; creating one *clause consistency gadget* $G_{\mathsf{h}}(\mathcal{C})$. The same holds for all occurrences of literals representing the same variable; creating n *variable consistency gadgets* $G_{\mathsf{h}}(v)$ for $v \in \mathcal{V}$. All occurrences of literals in a clause form a $K_{1,3}$ in G_{v}, where the clause vertex is the central vertex; creating m *satisfiability gadgets* $G_{\mathsf{v}}(c)$ for $c \in \mathcal{C}$. See Fig. 2 for an example.

Intuitively, the satisfiability gadgets allow us to encode local constraints on the literals for each clause. We use this to enforce "not-all-equal" satisfiability. By contrast, the consistency gadgets allow us to encode global constraints that span multiple clauses; i.e., relating literals that correspond to the same variable. This completes our construction of $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$; see Fig. 2 for an example.

Correctness. Lemmas 4 and 5 establish the correctness of our reduction. Hence, since our construction of $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$ requires $\Theta(m)$ time, USSVR recognition is NP-hard. Note that every USSVR can be redrawn on an $O(n^2) \times O(n^2)$ grid such that its visibilities are not changed by preserving the order of the endpoints in the x and y-projections of its unit squares [4]. This gives a certificate using polynomially-many bits that can be verified in polynomial time. Thus, USSVR recognition is NP-complete.

[2] $X_\Gamma(A) > X_\Gamma(B)$ means $X_\Gamma(a) > X_\Gamma(b)$ for all $a \in A$, $b \in B$

Lemma 4 *If $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$ has a USSVR, φ is satisfiable.*

Proof. Let Γ be a USSVR for $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$. We construct a truth assignment $\alpha \colon \mathcal{V} \to \{T, F\}$ as follows. For every variable $v \in \mathcal{V}$, we define

$$\alpha(v) = \begin{cases} T & \text{if } Y_\Gamma(G_{\mathsf{h}}(v)) \geq Y_\Gamma(G_{\mathsf{h}}(\mathcal{C})), \\ F & \text{otherwise.} \end{cases}$$

We claim that α satisfies φ. To see this, let us consider any clause vertex c for a clause $(\ell_1 \vee \ell_2 \vee \ell_3)$. Note that we distinguish duplicate literals by their order in φ as in Fig. 2. By construction, c is a cut vertex whose removal from G_{v} creates three components, each containing one literal vertex ℓ_i. Then by Lemma 1, for at least one such vertex, say ℓ_2, $X_\Gamma(\ell_2) \subseteq X_\Gamma(c)$. But in fact, since Γ is a unit-square representation, we have $X_\Gamma(\ell_2) = X_\Gamma(c)$. Hence, for every $k \in \{1, 3\}$, $X_\Gamma(\ell_k) \cap X_\Gamma(\ell_2) \neq \emptyset$. Applying Property 3, we see that $\Gamma(\ell_2)$ and $\Gamma(\ell_k)$ must not be both above or both below $\Gamma(c)$. Moreover, since every consistency gadget in our construction is a component of G_{h}, Lemma 2 implies that for all variables $v \in \mathcal{V}$, either $Y_\Gamma(G_{\mathsf{h}}(v)) \geq Y_\Gamma(G_{\mathsf{h}}(\mathcal{C}))$, or vice versa. Therefore, either $Y_\Gamma(\ell_2) \geq Y_\Gamma(c) \geq Y_\Gamma(\ell_k)$ for $k \in \{1, 3\}$, implying that α satisfies exactly one literal in c, or $Y_\Gamma(\ell_k) \geq Y_\Gamma(c) \geq Y_\Gamma(\ell_2)$, implying that α satisfies exactly two. By repeating this argument for all clauses in $\mathcal{C}$, we see that α satisfies φ. □

Lemma 5 *If φ is satisfiable, $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$ has a USSVR.*

Proof. Let $\alpha \colon \mathcal{V} \to \{T, F\}$ be a truth assignment satisfying φ. To construct a USSVR Γ for $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$, we first represent G_{v} and G_{h} as two sets of intervals on the x and y-axes respectively. The construction of these intervals is as follows.

For the ith clause $c = (\ell_1 \vee \ell_2 \vee \ell_3) \in \mathcal{C}$, since α satisfies exactly one or two of its literals, there must be one, say ℓ_2, that has a unique truth value. We represent both c and ℓ_2 on the x-axis by the interval $[3i+1, 3i+2]$. Moreover, assuming ℓ_1 and ℓ_3 are in order, we represent their corresponding literal vertices on the x-axis as the intervals $[3i, 3i+1]+\epsilon$ and $[3i+2, 3i+3]-\epsilon$ respectively, for some small $\epsilon > 0$.

Let $\rho \colon \mathcal{V} \cup \{\mathcal{C}\} \to \{0, \ldots, |\mathcal{V}|\}$ be a bijection satisfying $\rho(v) > \rho(\mathcal{C})$ if and only if $\alpha(v) = T$ for each $v \in \mathcal{V}$. For each variable consistency gadget $G_{\mathsf{h}}(v)$, we represent its vertices on the y-axis by the interval $[2\rho(v), 2\rho(v) + 1]$. We also represent the clause consistency gadget similarly, replacing $\rho(v)$ with $\rho(\mathcal{C})$.

Observe that each vertex in V is represented by two unit intervals, one on the x-axis and one on the y-axis. Thus, for each $u \in V$, we can define $\Gamma(u)$ to be the Cartesian product of its two corresponding intervals. To see that this gives a valid USSVR Γ for $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$, we make three observations.

1. Every gadget in G_{v} (resp., G_{h}) occupies a contiguous interval on the x-axis (resp., y-axis) that is disjoint from the intervals of other gadgets.
2. Every satisfiability gadget in G_{v} for a clause $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ is drawn such that $\Gamma(c)$ blocks vertical visibility between $\Gamma(\ell_2)$ and $\Gamma(\ell_k)$ for $k \in \{1, 3\}$ assuming ℓ_2 has the unique unique truth value of literals in c.
3. Every consistency gadget in G_{h} is drawn as a horizontal stack of unit squares (in order from left to right) that share a y-projection.

The first observation implies that no two gadgets in G_{v} (resp., G_{h}) share an (unwanted) visibility. The next two observations mean that the implied visibilities for each gadget in G_{v} and G_{h} are realized exactly. □

3.2 Generalizations

Notice that in the reduction given in Section 3.1, we make only the assumption that the x-projection of every allowable shape has the same size. Thus, we can adapt this reduction to any family of translates of shapes that share a fixed positive width. In Appendix C, we also prove that SVR recognition using rectangles is NP-complete. Again, we can adapt this reduction to any family of translates of shapes for which at least two have different widths; e.g., the family of L-shapes. These observations allow us to state the following.

Corollary 1 *Deciding if a pair of undirected graphs has an SVR using shapes from a family of translates of connected shapes with positive width and height is NP-hard.*

For families of translates of orthogonal polygonal paths with constant complexity (e.g., L-shapes), SVR recognition is also in NP; this follows by a similar argument to what we gave for USSVR recognition.

Corollary 2 *Deciding if a pair of undirected graphs has an SVR using shapes from a family of translates of orthogonal polygonal paths with constant complexity and positive width and height is NP-complete.*

4 Pairs of undirected paths

The hardness results of Section 3 utilize graphs that are not significantly more structurally complicated than paths. This motivates the question of whether pairs of (undirected) paths always admit SVRs and if not, whether there exists a polynomial time algorithm to decide when they do.

This question has an easy answer when the underlying shapes are rectangles. First, notice that given a pair of paths $\langle P_{\mathsf{v}} = (V, E_{\mathsf{v}}), P_{\mathsf{h}} = (V, E_{\mathsf{h}}) \rangle$ defined on the same vertex set, if $E_{\mathsf{v}} \cap E_{\mathsf{h}} \neq \emptyset$, then no RSVR or USSVR can exist because both x and y-projections of two rectangles or two squares cannot overlap unless the shapes themselves overlap. Otherwise, if $E_{\mathsf{v}} \cap E_{\mathsf{h}} = \emptyset$, perform the following:

> Algorithm A: For all $v \in V$, place $\Gamma(v)$ in the plane with its left corner at (i, j) where i (resp., j) is v's place along P_{v} (resp., P_{h}) from a fixed reference endpoint of the path; and set the side lengths of the rectangles to be $1 + \epsilon$ for a small $\epsilon > 0$.

It is easy to check that this satisfies all the visibilities. We remark that this algorithm was presented by Brass et al. [3] to compute a simultaneous embedding of two paths. This leads to the following observation: There exists an RSVR and USSVR of $\langle P_{\mathsf{v}}, P_{\mathsf{h}} \rangle$ if and only if $E_{\mathsf{v}} \cap E_{\mathsf{h}} = \emptyset$.

In fact, the result holds for any shapes that intersect if both of their x and y projections overlap. We turn our attention, therefore, to shapes that do not obey this property. Surprisingly, this question becomes significantly more complicated even for L-shapes which are simply the left and bottom sides of a rectangle. A *simultaneous visibility representation using fixed orientation L-shapes* $\{\llcorner\}$ (an LSVR) of $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$ is a pair $\langle \Gamma_{\mathsf{v}}, \Gamma_{\mathsf{h}} \rangle$ where Γ_{v} is a BVR, Γ_{h} is a BVR rotated 90° (with vertical bars and horizontal visibility), and for all $v \in V$, $\underline{y}_{\Gamma}(v) = \underline{x}_{\Gamma}(v)$ (i.e., $\Gamma_{\mathsf{v}}(v)$ and $\Gamma_{\mathsf{h}}(v)$ share their respective bottom and left endpoints).

4.1 LSVR of two undirected paths

Let $\langle P_{\mathsf{v}}, P_{\mathsf{h}} \rangle$ be two undirected paths defined on the same set of vertices $V = [n]$. By relabeling the vertices, we may assume that P_{h} is the path $(1, 2, \ldots, n)$ and P_{v} is $(\pi_1, \pi_2, \ldots, \pi_n)$ for a permutation π of $[n]$. While the paths are undirected, the algorithm considers 1 and π_1 to be the reference endpoints of P_{h} and P_{v}, respectively. We write $(\pi_i, \pi_{i+1}) \in P_{\mathsf{h}}$ but not $(\pi_{i+1}, \pi_i) \in P_{\mathsf{h}}$ because of this choice of reference endpoint. The intuition behind the following result may be understood by considering the result of running Algorithm A using L-shapes: $\Gamma(i)$ and $\Gamma(i+1)$ would intersect iff $(i+1, i) \in P_{\mathsf{v}}$—see Fig. 4. However, these intersecting L's may be modified to "nest" and preserve their existing visibilities in two circumstances (see Fig. 3). Notice that Algorithm A can produce drawings with four different "orders" by always using (i, j), $(-i, j)$, $(i, -j)$ or $(-i, -j)$ to place $\Gamma(v)$. Each of these would produce different drawings in

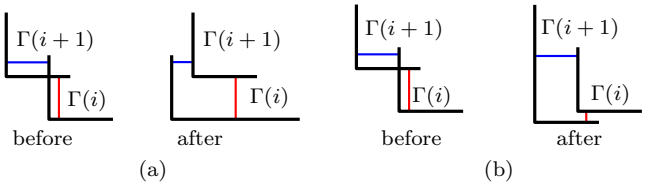


Figure 3: Two possible transformations to remove crossings. To perform (a), no other shape can have a visibility with $\Gamma(i+1)$ from below. To perform (b), no other shape can have a visibility with $\Gamma(i)$ from the left.

which the right ends of bars in Γ_{v} and the top ends of bars in Γ_{h} are either "increasing" or "decreasing". We show that if $\langle P_{\mathsf{v}}, P_{\mathsf{h}}\rangle$ admits any LSVR then modifying one of these four drawings to remove intersecting L's by nesting them will work. We also give four conditions, each of which forces the modification of one of the four drawings to fail.

In order to state the four conditions, we first need to define four subsets of the vertices. We say a sequence $S = (s_1, s_2, \ldots, s_k)$ is *increasing* (*decreasing*) in a sequence $T = (t_1, t_2, \ldots, t_n)$ if there exist indices $(j_1, j_2, \ldots, j_k)$ that are strictly increasing (decreasing) such that $s_i = t_{j_i}$ for all $i \in [k]$. A sequence S is *monotonic* in T if it is either increasing or decreasing in T. For example, $(4, 7, 3)$ is monotonic in $(1, 3, 2, 7, 5, 6, 4)$ but $(3, 4, 7)$ is not.

Definition 1 *For $P_{\mathsf{h}} = (1, 2, \ldots, n)$ and $P_{\mathsf{v}} = (\pi_1, \pi_2, \ldots, \pi_n)$ where π is a permutation of $[n]$, let*

1. *$A_\pi = (1, 2, \ldots, a)$ be the longest such sequence monotonic in π,*
2. *$B_\pi = (n, n-1, \ldots, b)$ be the longest such sequence monotonic in π,*
3. *$C_\pi = (\pi_1, \pi_2, \ldots, \pi_c)$ be the longest such sequence monotonic in $[n]$, and*
4. *$D_\pi = (\pi_n, \pi_{n-1}, \ldots, \pi_d)$ be the longest such sequence monotonic in $[n]$.*

For example, if $P_{\mathsf{v}} = (1, 3, 2, 7, 5, 6, 4)$, then $A_\pi = (1, 2)$, $B_\pi = (7, 6)$, $C_\pi = (1, 3)$, and $D_\pi = (4, 6)$. We are now ready to state the forbidden conditions. There exists i such that

F1. $(i+1, i) \in P_{\mathsf{v}}$, $i \notin C_\pi$, and $i+1 \notin A_\pi$;

F2. $(i, i+1) \in P_{\mathsf{v}}$, $i+1 \notin C_\pi$, and $i \notin B_\pi$;

F3. $(i, i+1) \in P_{\mathsf{v}}$, $i \notin D_\pi$, and $i+1 \notin A_\pi$;

F4. $(i+1, i) \in P_{\mathsf{v}}$, $i+1 \notin D_\pi$, and $i \notin B_\pi$.

The rest of Section 4.1 will focus on proving the following theorem.

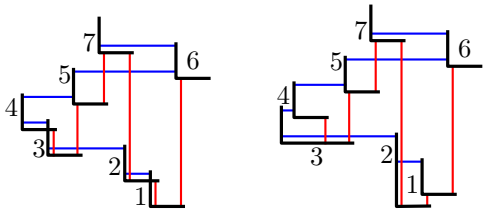


Figure 4: Left: The result of running Algorithm A on $P_{\mathsf{h}} = (1, 2, 3, 4, 5, 6, 7)$ and $P_{\mathsf{v}} = (4, 3, 5, 7, 2, 1, 6)$. Note the intersection of $\Gamma(4), \Gamma(3)$ and $\Gamma(2), \Gamma(1)$. Right: The intersections can be alleviated by stretching $\Gamma(3)$ and $\Gamma(2)$.

Theorem 6 *Let $\langle P_{\mathsf{v}}, P_{\mathsf{h}}\rangle$ be two paths defined on the same set of n vertices. There exists an LSVR of $\langle P_{\mathsf{v}}, P_{\mathsf{h}}\rangle$ if and only if at least one of conditions F1 through F4 is not met. Moreover, in the positive case, the LSVR is realizable in $O(n)$ time on a grid of size $O(n) \times O(n)$.*

Note that since each condition F1-F4 can be tested in linear time, Theorem 6 yields a linear time algorithm to determine if two given paths admit an LSVR. Before proceeding to the proof of Theorem 6, we present a technical result which will be a useful tool in many of the proofs to come. It demonstrates that once the x-projection of a bar is contained in that of another, this containment propagates for any representation of a path. Fig. 7b provides an example.

Lemma 7 *Let Γ be a noncollinear BVR of a path $P = v_1, \ldots, v_n$. If $X_\Gamma(v_j) \subset X_\Gamma(v_k)$ for $j < k$, then (i) $X_\Gamma(v_1) \subset X_\Gamma(v_2) \subset \cdots \subset X_\Gamma(v_j)$; and (ii) $\underline{y}_\Gamma(v_1), \ldots, \underline{y}_\Gamma(v_k)$ forms a strictly monotonic sequence.*

4.1.1 Necessity

We first prove if an LSVR of $\langle P_{\mathsf{v}}, P_{\mathsf{h}}\rangle$ exists, then at least one of conditions F1-F4 is not met. We begin by examining the structure of the underlying BVRs. For a path $P = (v_1, \ldots, v_n)$ and a BVR Γ of P, we say Γ is *monotonically increasing* (resp., decreasing) if $\overline{x}_\Gamma(v_j) < \overline{x}_\Gamma(v_{j+1})$ (resp., $\overline{x}_\Gamma(v_j) > \overline{x}_\Gamma(v_{j+1})$) for all $j \in [n-1]$. If Γ is monotonically increasing or decreasing we say it is *monotone*.[3] We say Γ is *strictly increasing* (resp., *decreasing*) if Γ is monotonically increasing (resp., decreasing) and $X_\Gamma(v_j) \not\subseteq X_\Gamma(v_{j+1})$ (resp., $X_\Gamma(v_{j+1}) \not\subseteq X_\Gamma(v_j)$) for all $j \in [n-1]$. For a BVR rotated by 90°, the same definitions apply with $\overline{y}_\Gamma(v)$ replacing $\overline{x}_\Gamma(v)$. A visibility representation in which no vertical or horizontal line contains the endpoints of two shapes from different vertices is called *noncollinear*. Finally, a BVR which is noncollinear and monotone is

[3] Note that monotonically increasing and decreasing are not symmetric properties: They both depend on the right side of the bars.

called *canonical*, and an LSVR $\Gamma = \langle \Gamma_\mathsf{v}, \Gamma_\mathsf{h} \rangle$ is *canonical* if Γ_v and Γ_h are both canonical.

We will apply the same definitions and notation to subdrawings of an LSVR Γ. That is, for a subset $S \subseteq V$, we will say $\Gamma(S)$ is monotone (or strictly increasing, etc) if the conditions are satisfied for the realizations of the vertices in S. The following two lemmas allows us to concentrate only on canonical LSVRs.

Lemma 8 *If $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$ has an LSVR then it has a canonical LSVR.*

Observe that in the transformations depicted in Fig. 3, $\Gamma_\mathsf{v}(\{i, i{+}1\})$ and $\Gamma_\mathsf{h}(\{i, i{+}1\})$ are altered from being strictly increasing to simply monotonically increasing. In order to determine when two L-shapes can be "uncrossed," therefore, we first determine in which parts of the drawing Γ_h and Γ_v are required to be strictly increasing or decreasing.

Lemma 9 *Suppose Γ is a canonical LSVR of $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$ and A_π, B_π, C_π and D_π are as in Definition 1. If Γ_v is monotonically increasing (resp., decreasing) then $\Gamma_\mathsf{v}(\{\pi_c, \pi_{c+1}, \ldots, \pi_n\})$ (resp., $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_{d-1}, \pi_d\})$) is strictly increasing (resp., decreasing). Similarly, if Γ_h is monotonically increasing (resp., decreasing) then $\Gamma_\mathsf{h}(\{a, a+1, \ldots, n\})$ (resp., $\Gamma_\mathsf{h}(\{1, \ldots, b-1, b\})$) is strictly increasing (resp., decreasing), where $a = |A_\pi|$, $b = n - |B_\pi| + 1$, $c = |C_\pi|$, and $d = n - |D_\pi| + 1$.*

We can now prove that if an LSVR of $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$ exists then at least one of the conditions F1-F4 are not met, which completes the proof of necessity.

By Lemma 8, we may assume that if an LSVR of $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$ exists then there is an LSVR Γ that is monotone and noncollinear. We claim that F1 prevents the existence of an LSVR Γ in which Γ_v and Γ_h are both monotonically increasing, F2 prevents Γ in which Γ_v is increasing and Γ_h is decreasing, F3 prevents Γ in which Γ_v is decreasing and Γ_h increasing, and F4 prevents Γ in which Γ_v and Γ_h are both decreasing. Since there are the only four possibilities for a monotone LSVR, by Lemma 8 if none of these four monotonic LSVRs exist, then no LSVR of $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$ exists. We provide the proof for the case of F1 only, as the other cases are argued similarly. Suppose Γ is an LSVR of $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$ in which Γ_v and Γ_h are monotonically increasing and let i be as in condition F1: $(i{+}1, i) \in P_\mathsf{v}$, $i \notin C_\pi$, and $i{+}1 \notin A_\pi$. Let $a = |A_\pi|$. By Lemma 9, $\Gamma_\mathsf{h}(\{a, a+1, \ldots, n\})$ is strictly increasing. Thus, since $i \geq a$ (because $i+1 \notin A_\pi$), $\underline{y}_\Gamma(i) < \underline{y}_\Gamma(i+1) < \overline{y}_\Gamma(i)$. By similar reasoning, we obtain that $\Gamma_\mathsf{v}(\{i+1, i\})$ is strictly increasing, so $\underline{x}_\Gamma(i+1) < \underline{x}_\Gamma(i) < \overline{x}_\Gamma(i+1)$. However, this is an impossible configuration to realize without an intersection between $\Gamma_\mathsf{h}(i)$ and $\Gamma_\mathsf{v}(i+1)$. Therefore no such LSVR exists.

4.1.2 Sufficiency

In this section we present an algorithm which constructs an LSVR for a pair of paths assuming that at least one of conditions F1-F4 is not met. For the sake of clarity, we will assume that F1 is not met. An explanation of how to modify the algorithm and the proofs for other conditions is in Appendix F.

LsvrPaths **Algorithm.** Let $\langle P_\mathsf{v} = (\pi_1, \pi_2, \ldots, \pi_n), P_\mathsf{h} = (1, 2, \ldots, n) \rangle$ be two paths defined on the same vertex set $[n]$. We break the algorithm into three steps.

Step 1: For all $i = 1, \ldots, n$, draw $\Gamma(\pi_i)$ such that its corner is at (i, π_i) and both bars have length $1 + \epsilon$. That is, $\underline{x}_\Gamma(\pi_i) = i$, $\underline{y}_\Gamma(\pi_i) = \pi_i$, $\overline{x}_\Gamma(\pi_i) = i + 1 + \epsilon$, and $\overline{y}_\Gamma(\pi_i) = \pi_i + 1 + \epsilon$. Note that this is Algorithm A.

Step 2: Let $C_\pi = \{\pi_1, \ldots, \pi_c\}$. If $\pi_1 > \pi_2 > \cdots > \pi_c$ and there are crossings in $\Gamma(C_\pi)$, then for all $\pi_i \in C_\pi$, stretch $\Gamma(\pi_i)$ to the left such that $\underline{x}_\Gamma(\pi_i) = 2 - i$.

Step 3: Let $A_\pi = \{1, \ldots, a\}$. If $(1, 2, \ldots, a)$ is decreasing in π and there are crossings in $\Gamma(A_\pi)$, then for all $i \in A_\pi \setminus C_\pi$, stretch $\Gamma(i)$ downwards such that $\underline{y}_\Gamma(i) = 2 - i$.

Observe that LsvrPaths requires linear time. Furthermore, the layout is contained in $[2 - n, n] \times [2 - n, n]$, i.e., a grid of size $O(n) \times O(n)$. Hence, the following lemma completes the proof of Theorem 6. The proof is given in Appendix E.

Lemma 10 *If $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$ are two paths defined on the same vertex set and condition F1 is not satisfied then Algorithm LsvrPaths returns an LSVR of $\langle P_\mathsf{v}, P_\mathsf{h} \rangle$.*

References

[1] T. Biedl, G. Liotta, and F. Montecchiani. On Visibility Representations of Non-Planar Graphs. In S. Fekete and A. Lubiw, editors, *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:16, 2016.

[2] P. Bose, A. Dean, J. Hutchinson, and T. Shermer. On rectangle visibility graphs. In *International Symposium on Graph Drawing*, pages 25–44, 1996.

[3] P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. P. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007.

[4] K. Casel, H. Fernau, A. Grigoriev, M. L. Schmid, and S. Whitesides. Combinatorial Properties and Recognition of Unit Square Visibility Graphs. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, 2017.

[5] F. J. Cobos, J. C. Dana, F. Hurtado, A. Márquez, and F. Mateos. On a visibility representation of graphs. In *International Symposium on Graph Drawing*, pages 152–161. Springer, 1995.

[6] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[7] A. M. Dean, W. S. Evans, E. Gethner, J. D. Laison, M. A. Safari, and W. T. Trotter. Bar k-visibility graphs. *J. Graph Algorithms Appl.*, 11(1):45–59, 2007.

[8] A. M. Dean and J. P. Hutchinson. Rectangle-visibility representations of bipartite graphs. In *International Symposium on Graph Drawing*, pages 159–166. Springer, 1994.

[9] A. M. Dean and N. Veytsel. Unit bar-visibility graphs. *Congressus Numerantium*, pages 161–176, 2003.

[10] E. Di. Giacomo, W. Didimo, W. S. Evans, G. Liotta, H. Meijer, F. Montecchiani, and S. K. Wismath. Orthopolygon visibility representations of embedded graphs. *Algorithmica*, 80(8):2345–2383, 2018.

[11] P. Duchet, Y. Hamidoune, M. Las Vergnas, and H. Meyniel. Representing a planar graph by vertical lines joining different levels. *Discrete Mathematics*, 46(3):319–321, 1983.

[12] A. Estrella-Balderrama, E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous geometric graph embeddings. In *International Symposium on Graph Drawing*, pages 280–290, 2007.

[13] W. S. Evans, G. Liotta, and F. Montecchiani. Simultaneous visibility representations of plane st-graphs using L-shapes. *Theoretical Computer Science*, 645:100–111, 2016.

[14] S. Felsner and M. Massow. Parameters of bar k-visibility graphs. *J. Graph Algorithms Appl.*, 12(1):5–27, 2008.

[15] S. G. Hartke, J. Vandenbussche, and P. Wenger. Further results on bar k-visibility graphs. *SIAM Journal on Discrete Mathematics*, 21(2):523–531, 2007.

[16] J. P. Hutchinson, T. Shermer, and A. Vince. On representations of some thickness-two graphs. *Computational Geometry*, 13(3):161–171, 1999.

[17] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.

[18] T. C. Shermer. On rectangle visibility graphs III. External visibility and complexity. In *Canadian Conference on Computational Geometry*, volume 96, pages 234–239, 1996.

[19] I. Streinu and S. Whitesides. Rectangle visibility graphs: characterization, construction, and compaction. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 26–37, 2003.

[20] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1(4):321–341, 1986.

[21] S. K. Wismath. Characterizing bar line-of-sight graphs. In *Proceedings of the 1st Annual ACM Symposium on Computational Geometry*, pages 147–152, 1985.

[22] S. K. Wismath. *Bar-Representable Visibility Graphs and Related Flow Problems.* PhD thesis, University of British Columbia, 1989.

A Omitted Proofs in Section 2

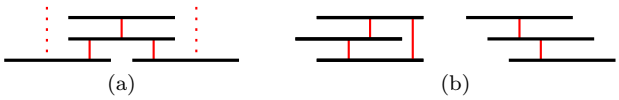
Let Γ be a (vertical) visibility representation for a graph $G = (V, E)$. The following properties hold.

Property 1 *For $S_1, S_2 \subset V$ and $x \in X_\Gamma(S_1) \cap X_\Gamma(S_2)$, there exists a path $u = u_1, \ldots, u_k = v$ in G for some $u \in S_1$ and $v \in S_2$ such that $x \in X_\Gamma(u_i)$ for all $i \in [k]$.*

Proof. Consider the intersection of Γ and the infinite vertical line $x \times (-\infty, +\infty)$. Since this line intersects both $\Gamma(S_1)$ and $\Gamma(S_2)$, there must be two vertices $u \in S_1$ and $v \in S_2$ such that $\Gamma(u)$ and $\Gamma(v)$ both intersect the line. Let $u = u_0, u_1, \ldots, u_k = v$ be the sequence of vertices in V that intersect the line in order along the line from $\Gamma(u)$ to $\Gamma(v)$. $\Gamma(u_i)$ and $\Gamma(u_{i+1})$ have an unblocked vertical visibility segment between them for all $1 \leq i < k$, which implies a path between u and v that connects S_1 and S_2 in G. □

Property 2 *If an endpoint of $X_\Gamma(w)$ is strictly contained in $X_\Gamma(u) \cap X_\Gamma(v)$ and $\underline{y}_\Gamma(u) < \underline{y}_\Gamma(w) < \underline{y}_\Gamma(v)$ for $u, v, w \in V$, then there is a cycle in G.*

Proof. Since an endpoint of $X_\Gamma(w)$ is strictly contained in $X_\Gamma(u) \cap X_\Gamma(v)$, there exists $x \in X_\Gamma(w)$ and $x' \notin X_\Gamma(w)$ such that $x, x' \in X_\Gamma(u) \cap X_\Gamma(v)$. By Property 1, there exists a path from u to v (following the vertical line through x) that, since $\underline{y}_\Gamma(u) < \underline{y}_\Gamma(w) < \underline{y}_\Gamma(v)$, contains w, and a path from u to v (following the vertical line through x') that does not contain w. The union of these two paths contains a cycle. □



(a) (b)

Figure 5: Illustrations of Lemma 1 and 2 (Fig. 5a and 5b respectively). Lemma 1 says that a cut vertex will not *nest*[4] at most two components induced by its removal. Lemma 2 states that disjoint subgraphs must not overlap.

Property 3 *Let $u_1, \ldots, u_\ell$ be the only path from u_1 to u_ℓ in G. If $\Gamma(u_i)$ and $\Gamma(u_k)$ are both above or both below $\Gamma(u_j)$ for $i < j < k$, then $X_\Gamma(\{u_1, \ldots, u_i\}) \cap X_\Gamma(\{u_k, \ldots, u_\ell\}) = \emptyset$.*

Proof. If $x \in X_\Gamma(\{u_1, \ldots, u_i\}) \cap X_\Gamma(\{u_k, \ldots, u_\ell\})$, then by Property 1, there exists a path from u_i to u_k (following the vertical line through x) that, since $\Gamma(u_i)$ and $\Gamma(u_k)$ are both above or both below $\Gamma(u_j)$, does not include u_j, a contradiction. □

B Omitted Proofs in Section 3

Lemma 1 *Let $G = (V, E)$ be a connected graph with a (vertical) visibility representation Γ. If $u \in V$ is a cut vertex whose removal creates components $C_1, \ldots, C_k$ then $X_\Gamma(C_i) \not\subseteq X_\Gamma(u)$ for at most two components.*

Proof. Since G is connected, $X_\Gamma(C_i)$ intersects $X_\Gamma(u)$ and if in addition $X_\Gamma(C_i) \not\subseteq X_\Gamma(u)$, then since C_i is connected, $X_\Gamma(C_i)$ is a contiguous interval that strictly contains an endpoint of $X_\Gamma(u)$. If three components have this property then for two of them, say C_i and C_j, $X_\Gamma(C_i)$ and $X_\Gamma(C_j)$ strictly contain the same endpoint and thus contain a point $x \notin X_\Gamma(u)$. By Property 1, G contains a path (following the vertical line through x) between C_i and C_j that does not contain u, a contradiction. □

Lemma 2 *Let $G = (V, E)$ be a graph with a (vertical) visibility representation Γ. If C_1 and C_2 are components in G, then either $X_\Gamma(C_1) > X_\Gamma(C_2)$[5], or vice versa.*

Proof. Since each component C_i is connected, its x-projection $X_\Gamma(C_i)$ forms a contiguous interval. And yet since C_1 and C_2 are disconnected in G, by Property 1, $X_\Gamma(C_1) \cap X_\Gamma(C_2) = \emptyset$. Thus either $X_\Gamma(C_1) > X_\Gamma(C_2)$, or vice versa. □

See Fig. 5 for illustrations of Lemma 1 and 2.

C Hardness of RSVR recognition

In this section, we prove that determining if a pair of undirected graphs has a *simultaneous visibility representation using rectangles* (RSVR) is NP-complete. In contrast to the proof given in Section 3.1, here, we reduce from the NP-complete problem of 3SAT [6]. A new reduction is needed since every pair of edge-disjoint caterpillar forests (as produced for the reduction in Section 3.1) has an RSVR due to Theorem 5 by Bose et al. [2].

Our modified construction is not much more complicated than before: one graph remains a set of disjoint paths while the other is a set of disjoint trees with 3 leaves connected to a root by paths of length 2. This slight modification allows us to prove the following theorem.

Theorem 11 *Deciding if a pair of undirected graphs has an RSVR is NP-complete.*

Construction. Let φ be an instance of 3SAT with a set $\mathcal{C}$ of m clauses and a set $\mathcal{V}$ of $n \leq 3m$ variables.

We adapt the gadgets used in Section 3.1 to this setting as follows. Each satisfiability gadget $G_\mathsf{v}(c)$ for $c \in \mathcal{C}$ is now a 1-subdivision of $K_{1,3}$ (i.e. $\sqrt{\ }$) where the central vertex is the clause c, the subdivision vertices are the occurrences of literals in the clause, and each leaf is an occurrence of the negation of its parent. In addition to the variable consistency gadget, we also construct a *negated variable consistency gadget* $G_\mathsf{h}(\overline{v})$ for each variable $v \in \mathcal{V}$ that is the path of negated occurrences of literals in the order of their appearance in φ. This completes our construction of $\langle G_\mathsf{v}, G_\mathsf{h} \rangle$; see Fig. 6 for an example.

Correctness. Lemmas 12 and 13 establish the correctness of our reduction. Thus, by a similar argument to the one found in Section 3.1, RSVR recognition is NP-complete.

Lemma 12 *If $\langle G_\mathsf{v}, G_\mathsf{h} \rangle$ has an RSVR, φ is satisfiable.*

Proof. Let Γ be an RSVR for $\langle G_\mathsf{v}, G_\mathsf{h} \rangle$. If, for some variable $v \in \mathcal{V}$, we have $Y_\Gamma(G_\mathsf{h}(\overline{v})) \geq Y_\Gamma(G_\mathsf{h}(v)) \geq Y_\Gamma(G_\mathsf{h}(\mathcal{C}))$, or vice versa, we say that v is *positively-arranged* in Γ. We construct a truth assignment $\alpha \colon \mathcal{V} \to \{T, F\}$ as follows. For each variable $v \in \mathcal{V}$, we define

$$\alpha(v) = \begin{cases} T & \text{if } v \text{ is positively-arranged in } \Gamma, \\ F & \text{otherwise.} \end{cases}$$

We claim that α satisfies φ. To see this, let us consider any clause vertex c for a clause $(\ell_1 \vee \ell_2 \vee \ell_3)$. By construction, c is a cut vertex whose removal from $G_\mathsf{v}(c)$ creates three components, each containing one literal vertex and its negated counterpart. Then by Lemma 1, for at least one literal, say ℓ_2, we have $X_\Gamma(\{\ell_2, \overline{\ell_2}\}) \subseteq X_\Gamma(c)$. Hence, $X_\Gamma(\overline{\ell_2}) \cap X_\Gamma(c) \neq \emptyset$.

Applying Property 3, we see that $\Gamma(\overline{\ell_2})$ and $\Gamma(c)$ must not be both above or both below $\Gamma(\ell_2)$. Moreover, since the consistency gadgets in our construction are components of G_h, Lemma 2 implies that their y-projections must form disjoint intervals. Therefore, either $Y_\Gamma(\overline{\ell_2}) \geq Y_\Gamma(\ell_2) \geq Y_\Gamma(c)$, or vice versa. Thus, if ℓ_2 is a positive literal then its variable is positively-arranged in Γ; otherwise, ℓ_2 is a negative literal implying that its variable is not positively-arranged in Γ. In either case, α satisfies c. By repeating this argument for all clauses in $\mathcal{C}$, we see that α satisfies φ. □

Lemma 13 *If φ is satisfiable, $\langle G_\mathsf{v}, G_\mathsf{h} \rangle$ has an RSVR.*

[4] $\Gamma(A)$ is *nested* in $\Gamma(B)$ if $X_\Gamma(A) \supseteq X_\Gamma(B)$

[5] $X_\Gamma(A) > X_\Gamma(B)$ means $X_\Gamma(a) > X_\Gamma(b)$ for all $a \in A$, $b \in B$

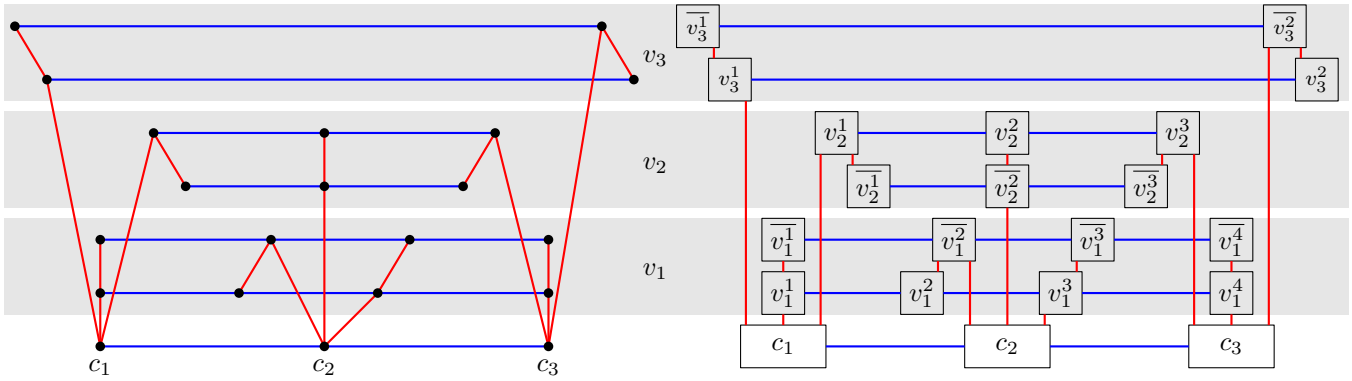


Figure 6: Left: G_{v} (red) and G_{h} (blue) for 3SAT instance $\varphi = (v_3 \vee v_1 \vee v_2)(\overline{v_1} \vee \overline{v_2} \vee v_1)(v_2 \vee v_1 \vee \overline{v_3})$. Right: An RSVR of $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$ encoding the truth assignment $v_1, v_3 = T$ and $v_2 = F$.

Proof. Let $\alpha \colon \mathcal{V} \to \{T, F\}$ be a truth assignment satisfying φ. To construct an RSVR Γ for $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$, we first represent G_{v} and G_{h} as two sets of intervals on the x and y-axes respectively. The construction of these intervals is as follows.

For the ith clause $c = (\ell_1 \vee \ell_2 \vee \ell_3) \in \mathcal{C}$, we represent c on the x-axis by the interval $[7i+2, 7i+5]$. Next, for one of the satisfied literals in c, say ℓ_2, we represent both ℓ_2 and $\overline{\ell_2}$ on the x-axis by the interval $[7i+3, 7i+4]$. Finally, assuming that ℓ_1 and ℓ_3 are in order, we represent ℓ_1 and $\overline{\ell_1}$ on the x-axis by the intervals $[7i+1, 7i+2] + \epsilon$ and $[7i, 7i+1] + 2\epsilon$ respectively, for some positive but small ϵ. Similarly, we represent ℓ_3 and $\overline{\ell_3}$ by the intervals $[7i+5, 7i+6] - \epsilon$ and $[7i+6, 7i+7] - 2\epsilon$ respectively.

For the jth variable $v \in \mathcal{V}$, if $\alpha(v) = T$, we represent the vertices in $G_{\mathsf{h}}(v)$ and $G_{\mathsf{h}}(\overline{v})$ on the y-axis by the intervals $[4j, 4j+1]$ and $[4j+2, 4j+3]$. Otherwise, if $\alpha(v) = F$, we simply swap the intervals and proceed as before. Finally, we represent every vertex in $G_{\mathsf{h}}(\mathcal{C})$ on the y-axis by the interval $[0, 1]$.

Observe that each vertex in V is represented by two (nonempty) intervals, one on the x-axis and one on the y-axis. Thus, for every $u \in V$, we can define $\Gamma(u)$ to be the Cartesian product of its two corresponding intervals. To see this gives a valid RSVR Γ for $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$, we make three observations.

1. Every gadget in G_{v} (resp., G_{h}) occupies a contiguous interval on the x-axis (resp., y-axis) that is disjoint from the intervals of other gadgets.
2. Every satisfiability gadget in G_{v} for a clause $c = (\ell_1 \vee \ell_2 \vee \ell_3)$ is drawn such that $\Gamma(\ell_2)$ blocks vertical visibility between $\Gamma(c)$ and $\Gamma(\overline{\ell_2})$. Moreover, $X_\Gamma(c)$ intersects $X_\Gamma(\ell_k)$ but not $X_\Gamma(\overline{\ell_k})$ for $k \in \{1, 3\}$.
3. Every consistency gadget in G_{h} is drawn as a horizontal stack of rectangles (in order from left to right) that share a y-projection.

The first observation implies that no two gadgets in G_{v} (resp., G_{h}) share an (unwanted) visibility. The next two observations mean that the implied visibilities for each gadget in G_{v} and G_{h} are realized exactly. Therefore, Γ is indeed a valid RSVR for $\langle G_{\mathsf{v}}, G_{\mathsf{h}} \rangle$. □

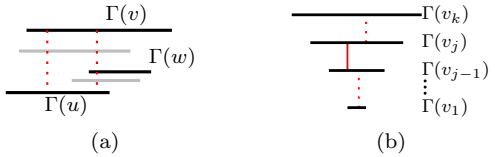


Figure 7: Illustration of Property 2 and Lemma 7 (Fig. 7a and 7b respectively). The gray lines in (a) represent other bars which may or may not be in the BVR, and the dotted (red) lines represent the two sequences of vertical visibilities.

D Omitted Proofs for Section 4

Lemma 7 *Let Γ be a noncollinear BVR of a path $P = v_1, \ldots, v_n$. If $X_\Gamma(v_j) \subset X_\Gamma(v_k)$ for $j < k$, then (i) $X_\Gamma(v_1) \subset X_\Gamma(v_2) \subset \cdots \subset X_\Gamma(v_j)$; and (ii) $\underline{y}_\Gamma(v_1), \ldots, \underline{y}_\Gamma(v_k)$ forms a strictly monotonic sequence.*

Proof. By assumption we have that $X_\Gamma(v_j) \subset X_\Gamma(v_k)$, and since Γ is noncollinear, we may assume that $\underline{y}_\Gamma(v_j) < \underline{y}_\Gamma(v_k)$ or $\underline{y}_\Gamma(v_j) > \underline{y}_\Gamma(v_k)$. Suppose it is the former; the argument is symmetric in the other case. Consider the largest $i < j$ such that either $X_\Gamma(v_i) \not\subset X_\Gamma(v_{i+1})$ or $\underline{y}_\Gamma(v_i) > \underline{y}_\Gamma(v_{i+1})$. If $\underline{y}_\Gamma(v_i) > \underline{y}_\Gamma(v_{i+1})$, then $\Gamma(v_i)$ and $\Gamma(v_k)$ are both above $\Gamma(v_{i+1})$. By Property 3, $X_\Gamma(\{v_1, \ldots, v_i\}) \cap X_\Gamma(\{v_k, \ldots, v_n\}) = \emptyset$. However, since $X_\Gamma(v_i) \cap X_\Gamma(v_{i+1}) \neq \emptyset$ (they must share a visibility) and $X_\Gamma(v_{i+1}) \subset X_\Gamma(v_k)$, it follows that $X_\Gamma(v_i) \cap X_\Gamma(v_k) \neq \emptyset$, a contradiction. Otherwise, if $X_\Gamma(v_i) \not\subset X_\Gamma(v_{i+1})$, then one of the endpoints of $X_\Gamma(v_{i+1})$ is contained in $X_\Gamma(v_i)$ and $X_\Gamma(v_k)$. By Property 2, there is a cycle, a contradiction. □

Lemma 8 *If $\langle P_{\mathsf{v}}, P_{\mathsf{h}} \rangle$ has an LSVR then it has a canonical LSVR.*

Proof. We begin by demonstrating that:

Claim 1 *If $\langle P_{\mathsf{v}}, P_{\mathsf{h}} \rangle$ has an LSVR then it has a noncollinear LSVR.*

Proof. We first show how to transform an LSVR Γ of $\langle P_\mathsf{v}, P_\mathsf{h}\rangle$ into an LSVR Γ' such that $\overline{x}_{\Gamma'}(i) \neq \underline{x}_{\Gamma'}(j)$ and $\underline{y}_{\Gamma'}(i) \neq \overline{y}_{\Gamma'}(j)$ for all $i, j \in V$. Suppose that $\overline{x}_\Gamma(i) = \underline{x}_\Gamma(j) = x_0$ for some $i, j \in V$. Let $L = \{k \in V : \overline{x}_\Gamma(k) \leq x_0\}$ and $R = \{k \in V : \underline{x}_\Gamma(k) \geq x_0\}$ be the collection of shapes to the left and right of x_0 respectively. Let $\delta > 0$. Construct Γ' as follows.

Shift $\Gamma(L)$ to the left by δ, and $\Gamma(R)$ to the right by δ. For all $\ell \notin L \cup R$ (meaning that $\underline{x}_\Gamma(\ell) < x_0 < \overline{x}_\Gamma(\ell)$), stretch $\Gamma(\ell)$ both left and right by δ (so that $\underline{x}_{\Gamma'}(\ell) = \underline{x}_\Gamma(\ell) - \delta$ and $\overline{x}_{\Gamma'}(\ell) = \overline{x}_\Gamma(\ell) + \delta$). Note that $\overline{x}_{\Gamma'}(i) < \underline{x}_{\Gamma'}(j)$.

We claim that Γ' is an LSVR of $\langle P_\mathsf{v}, P_\mathsf{h}\rangle$. Since there was no vertical displacement, the horizontal visibilities present in Γ were unaffected. Moreover, the structure of the drawing to the right of (and including) x_0 in Γ was unchanged in Γ'. Formally, $\Gamma' \cap [x_0 + \delta, \infty) \times (-\infty, \infty)$ is precisely $\Gamma \cap [x_0, \infty) \times (-\infty, \infty)$ shifted by δ. Similarly for the structure to the left of x_0 in Γ. Since $\Gamma'_\mathsf{h}(v)$ lies outside the region $M = (x_0 - \delta, x_0 + \delta) \times (-\infty, \infty)$ for all $v \in V$, we introduce no crossings. Therefore, it remains only to show that no unwanted vertical visibilities are introduced in M. Notice that any unwanted vertical visibilities in this region must be among shapes not in $\Gamma'(L \cup R)$ (i.e., those which were stretched), since $\Gamma'(L \cup R) \cap M = \emptyset$.

If $\Gamma'_\mathsf{v}(u)$ and $\Gamma'_\mathsf{v}(v)$ share an unwanted vertical visibility in M then, since the visibility was blocked by two horizontal bars which shared a collinearity at x_0 in Γ—say $\Gamma_\mathsf{v}(i')$ and $\Gamma_\mathsf{v}(j')$—$X_\Gamma(u)$ and $X_\Gamma(v)$ both strictly contain x_0 and $\Gamma_\mathsf{v}(u)$ is above $\Gamma_\mathsf{v}(i')$ and $\Gamma_\mathsf{v}(j')$ while $\Gamma_\mathsf{v}(v)$ is below them. However, this implies a cycle by Property 2, a contradiction. This demonstrates that Γ' is an LSVR of $\langle P_\mathsf{v}, P_\mathsf{h}\rangle$. If instead $\underline{x}_{\Gamma'}(i) = \overline{y}_{\Gamma'}(j)$, then we perform a similar surgery, but the geometry is rotated by $\pi/2$. Repeating this process iteratively produces the desired LSVR of $\langle P_\mathsf{v}, P_\mathsf{h}\rangle$. We now assume that Γ is an LSVR such that $\overline{x}_\Gamma(i) \neq \underline{x}_\Gamma(j)$ and $\underline{y}_\Gamma(i) \neq \overline{y}_\Gamma(j)$ for all $i, j \in V$, and show how to transform Γ into an LSVR Γ' such that $\underline{y}_{\Gamma'}(i) \neq \underline{y}_{\Gamma'}(j)$, $\overline{y}_{\Gamma'}(i) \neq \overline{y}_{\Gamma'}(j)$, $\underline{x}_{\Gamma'}(i) \neq \underline{x}_{\Gamma'}(j)$, and $\overline{x}_{\Gamma'}(i) \neq \overline{x}_{\Gamma'}(j)$ for all $i, j \in V$. Suppose $\overline{x}_\Gamma(i) = \overline{x}_\Gamma(j) = x_0$ for some $i, j \in V$ (as above, the other cases are symmetric) and choose i and j such that $\Gamma_\mathsf{v}(i)$ is the highest such bar and $\Gamma_\mathsf{v}(j)$ is the lowest. We construct an LSVR Γ' which removes this collinearity. By Property 2, there cannot be a bar $\Gamma_\mathsf{v}(u)$ above $\Gamma_\mathsf{v}(i)$ and a bar $\Gamma_\mathsf{v}(v)$ below $\Gamma_\mathsf{v}(j)$ that both strictly contain x_0. If such a $\Gamma_\mathsf{v}(u)$ does not exist, decrease $\overline{x}_\Gamma(i)$ by a small amount. This does not introduce any new vertical visibility since $\Gamma_\mathsf{v}(i)$ is the highest bar with $\overline{x}_\Gamma(i) = x_0$. Similarly, decrease $\overline{x}_\Gamma(j)$ if such a $\Gamma_\mathsf{v}(v)$ does not exist. Repeating this process iteratively produces a noncollinear LSVR of $\langle P_\mathsf{v}, P_\mathsf{h}\rangle$. □

We now proceed to the main proof of Lemma 8. By Lemma 1, we may assume that Γ is noncollinear. Suppose without loss of generality that $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_{k-1}\})$ is monotone increasing for some $k \geq 3$. If $\overline{x}_\Gamma(\pi_k) < \overline{x}_\Gamma(\pi_{k-1})$, we will show how to modify Γ such that $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_k\})$ is monotonically increasing or decreasing. We consider three cases:

Case 1: $\Gamma_\mathsf{v}(\pi_k)$ and $\Gamma_\mathsf{v}(\pi_{k-2})$ are both above or both below $\Gamma_\mathsf{v}(\pi_{k-1})$, and $\overline{x}_\Gamma(\pi_{k-2}) < \underline{x}_\Gamma(\pi_k)$. There is no vertex u such that $\overline{x}_\Gamma(\pi_k) \in X_\Gamma(u)$ as otherwise, by Property 1, there is a path from u to π_{k-1} avoiding π_k and π_{k-2}, implying that π_{k-1} has degree three in P_v; a contradiction. Hence, we may stretch $\Gamma_\mathsf{v}(\pi_k)$ to the right such that $\overline{x}_\Gamma(\pi_k) > \overline{x}_\Gamma(\pi_{k-1})$, making $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_k\})$ monotonically increasing.

Case 2: $\Gamma_\mathsf{v}(\pi_k)$ and $\Gamma_\mathsf{v}(\pi_{k-2})$ are both above or both below $\Gamma_\mathsf{v}(\pi_{k-1})$, and $\overline{x}_\Gamma(\pi_k) < \underline{x}_\Gamma(\pi_{k-2})$. Since $\Gamma_\mathsf{v}(\pi_k)$ and $\Gamma_\mathsf{v}(\pi_{k-1})$ must share a visibility, we have that $\underline{x}_\Gamma(\pi_{k-1}) < \underline{x}_\Gamma(\pi_{k-2})$, hence $X_\Gamma(\pi_{k-2}) \subset X_\Gamma(\pi_{k-1})$ By Lemma 7(i), $X_\Gamma(\pi_1) \subset X_\Gamma(\pi_2) \subset \cdots \subset X_\Gamma(\pi_{k-1})$. Successively, for j from $k-2$ to 1, stretch $\Gamma_\mathsf{v}(\pi_j)$ to the right such that $\overline{x}_\Gamma(\pi_j) > \overline{x}_\Gamma(\pi_{j+1})$. Note that before each stretch, $\overline{x}_\Gamma(\pi_{j+1})$ is the rightmost point in $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_k\})$; otherwise, as in Case 1, π_{j+1} has degree three. Thus, the transformation induces no unwanted vertical visibilities. Furthermore, it's clear that the horizontal visibilities are maintained since there is no movement of any vertical bars. $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_k\})$ is now monotonically decreasing, which completes the proof of this case.

Case 3: If one of $\Gamma_\mathsf{v}(\pi_k)$ and $\Gamma_\mathsf{v}(\pi_{k-2})$ is below $\Gamma_\mathsf{v}(\pi_{k-1})$ and the other above then, stretch $\Gamma_\mathsf{v}(\pi_k)$ to the right such that $\overline{x}_\Gamma(\pi_k) > \overline{x}_\Gamma(\pi_{k-1})$. The argument that this induces no unwanted visibilities is the same as in Case 1. This completes the proof if $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_{k-1}\}$ is monotonically increasing. If $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_{k-1}\})$ is monotonically decreasing, Property 2 (for π_k, π_{k-1}, and π_{k-2}) implies a cycle if $\overline{x}_\Gamma(\pi_k) > \overline{x}_\Gamma(\pi_{k-1})$. Therefore, $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_k\})$ is monotonically decreasing. A similar and symmetric argument may be applied to Γ_h. □

Lemma 9 *Suppose Γ is a canonical LSVR of $\langle P_\mathsf{v}, P_\mathsf{h}\rangle$ and A_π, B_π, C_π and D_π are as in Definition 1. If Γ_v is monotonically increasing (resp., decreasing) then $\Gamma_\mathsf{v}(\{\pi_c, \pi_{c+1}, \ldots, \pi_n\})$ (resp., $\Gamma_\mathsf{v}(\{\pi_1, \ldots, \pi_{d-1}, \pi_d\})$) is strictly increasing (resp., decreasing). Similarly, if Γ_h is monotonically increasing (resp., decreasing) then $\Gamma_\mathsf{h}(\{a, a+1, \ldots, n\})$ (resp., $\Gamma_\mathsf{h}(\{1, \ldots, b-1, b\})$) is strictly increasing (resp., decreasing), where $a = |A_\pi|$, $b = n - |B_\pi| + 1$, $c = |C_\pi|$, and $d = n - |D_\pi| + 1$.*

Proof. Let Γ be a canonical BVR of a path $P = v_1, \ldots, v_n$. We begin by showing:

Claim 2 *If $\Gamma(\{v_i, v_{i+1}\})$ is strictly increasing (resp., decreasing) then $\Gamma(\{v_i, v_{i+1}, \ldots, v_n\})$ is strictly increasing (resp., decreasing).*

Proof. Suppose that $\Gamma(\{v_i, v_{i+1}\})$ is strictly increasing (the case of strictly decreasing is similar) and let $j > i$ be minimal such that $\Gamma(\{v_j, v_{j+1}\})$ is not. Since Γ is monotone and noncollinear, this implies $\underline{x}_\Gamma(v_{j+1}) < \underline{x}_\Gamma(v_j)$ and $X_\Gamma(v_j) \subset X_\Gamma(v_{j+1})$. Suppose that $\Gamma(v_{j-1})$ is above $\Gamma(v_j)$; the other case is argued similarly. If $\Gamma(v_{j+1})$ is above $\Gamma(v_{j-1})$ or below $\Gamma(v_j)$, applying Property 2 to v_{j-1}, v_j and v_{j+1} yields a contradiction. On the other hand, if $\Gamma(v_{j+1})$ is between $\Gamma(v_{j-1})$ and $\Gamma(v_j)$, then because $X_\Gamma(v_j) \subset X_\Gamma(v_{j+1})$, $\Gamma(v_{j-1})$ and $\Gamma(v_j)$ cannot share a visibility. This is a contradiction. □

We now show that:

Claim 3 *If Γ is a canonical LSVR of $\langle P_\mathsf{v}, P_\mathsf{h}\rangle$ and Γ_v is monotonically increasing (resp., decreasing) then*

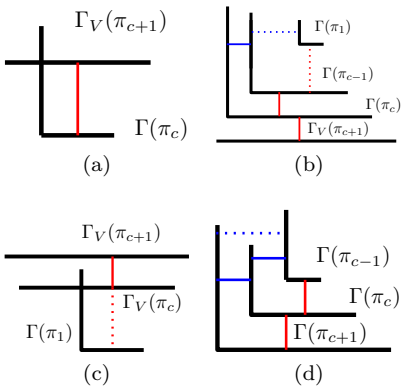


Figure 8: The cases in the proof of Lemma 9. (a) and (b) correspond to the two subcases of Case 1; (b) and (d) to the two subcases of Case 2.

$\Gamma_v(\{\pi_c, \pi_{c+1}\})$ (resp., $\Gamma_v(\{\pi_{d-1}, \pi_d\})$) is strictly increasing (resp., decreasing). Similarly, if Γ_h is monotonically increasing (resp., decreasing) then $\Gamma_h(\{a, a+1\})$ (resp., $\Gamma_h(\{b-1, b\})$) is strictly increasing (resp., decreasing).

Proof. Consider Γ_v, the proof for Γ_h can be obtained by a symmetric argument. We will assume that both Γ_v and Γ_h are monotonically increasing. If Γ_h is decreasing then the proofs pertaining to the two cases below are simply exchanged. Assume for contradiction that $\Gamma_v(\{\pi_c, \pi_{c+1}\})$ is not strictly increasing, so $\underline{x}_\Gamma(\pi_{c+1}) < \underline{x}_\Gamma(\pi_c)$. Notice that, since Γ_v is monotonically increasing, $X_\Gamma(\pi_c) \subset X_\Gamma(\pi_{c+1})$. Applying Lemma 7(i) we see that $X_\Gamma(\pi_i) \subset X_\Gamma(\pi_{i+1})$ for $i \in [c]$. We consider two cases based on the ordering of $(\pi_1, \ldots, \pi_c)$ in P_h (see Figure 8).

Case 1: Suppose $\pi_1 < \pi_2 < \cdots < \pi_c$. Then $\overline{y}_\Gamma(\pi_1) < \overline{y}_\Gamma(\pi_2) < \cdots < \overline{y}_\Gamma(\pi_c)$ since $(\pi_1, \pi_2, \ldots, \pi_c)$ is increasing in P_h and Γ_h is monotonically increasing. By definition of C_π, $\pi_{c+1} < \pi_c$, hence $\overline{y}_\Gamma(\pi_{c+1}) < \overline{y}_\Gamma(\pi_c)$. Therefore, $\Gamma_v(\pi_c)$ is above $\Gamma_v(\pi_{c+1})$ (i.e., $\underline{y}_\Gamma(\pi_c) > \underline{y}_\Gamma(\pi_{c+1})$); otherwise $\Gamma_h(\pi_c)$ would intersect $\Gamma_v(\pi_{c+1})$ (by the assumption that $\underline{x}_\Gamma(\pi_{c+1}) < \underline{x}_\Gamma(\pi_c)$). Hence, by Lemma 7(ii), $\underline{y}_\Gamma(\pi_1) > \underline{y}_\Gamma(\pi_2) > \cdots > \underline{y}_\Gamma(\pi_{c+1})$ thus demonstrating that $\Gamma(\pi_i)$ is nested in $\Gamma(\pi_{i+1})$ for all $i \in [c-1]$.

First, we claim that for all $i \in \{\pi_1, \pi_1 + 1, \ldots, \pi_c\}$, $\underline{x}_\Gamma(i) \in [\underline{x}_\Gamma(\pi_c), \underline{x}_\Gamma(\pi_1)]$. Suppose the claim is false and choose the smallest $i \in \{\pi_1, \pi_1 + 1, \ldots, \pi_c\}$ such that either $\underline{x}_\Gamma(i) < \underline{x}_\Gamma(\pi_c)$ or $\underline{x}_\Gamma(i) > \underline{x}_\Gamma(\pi_1)$. In the former case, $\Gamma_h(i)$ is blocked from sharing a visibility with $\Gamma_h(i-1)$ by $\Gamma_h(\pi_c)$ (where we're using that $\overline{y}_\Gamma(i-1), \overline{y}_\Gamma(i) < \overline{y}_\Gamma(\pi_c)$, and $\underline{y}_\Gamma(i-1) > \underline{y}_\Gamma(\pi_c)$ to avoid $\Gamma_h(i-1)$ and $\Gamma_v(\pi_c)$ intersecting). In the latter case apply Property 2 to i, π_1 and π_c (using that $\overline{y}_\Gamma(\pi_c) > \overline{y}_\Gamma(i) > \overline{y}_\Gamma(\pi_1)$) to obtain a contradiction. This proves the claim, which implies that $\pi_{c+1} < \pi_1$, since $\pi_{c+1} < \pi_c$ and $\underline{x}_\Gamma(\pi_{c+1}) \notin [\underline{x}_\Gamma(\pi_c), \underline{x}_\Gamma(\pi_1)]$. This, however, also yields a contradiction: Since $Y_\Gamma(\pi_1) \subset Y_\Gamma(\pi_c)$ and $\underline{x}_\Gamma(\pi_c) < \underline{x}_\Gamma(\pi_1)$, Lemma 7 implies that $\underline{x}_\Gamma(i) > \underline{x}_\Gamma(\pi_c)$ for all $i < \pi_1$. By assumption however, $\underline{y}_\Gamma(\pi_{c+1}) < \underline{y}_\Gamma(\pi_c)$.

Case 2: Otherwise, $\pi_1 > \pi_2 > \cdots > \pi_c$ so $\overline{y}_\Gamma(\pi_1) > \overline{y}_\Gamma(\pi_2) > \cdots > \overline{y}_\Gamma(\pi_c)$. If $\underline{y}_\Gamma(\pi_{c+1}) < \underline{y}_\Gamma(\pi_c)$, then we must have $c = 1$; otherwise apply Property 2 to π_{c+1}, π_{c-1} and π_c. This, however, contradicts the definition of C_π as it will always have length at least two. Thus $\underline{y}_\Gamma(\pi_c) < \underline{y}_\Gamma(\pi_{c+1})$ and again by Lemma 7(ii) we have $\underline{y}_\Gamma(\pi_1) < \cdots < \underline{y}_\Gamma(\pi_{c+1})$. Again, however, this forces $c = 1$. Otherwise, because $\underline{x}_\Gamma(\pi_c) < \underline{x}_\Gamma(\pi_1)$ and $\overline{y}_\Gamma(\pi_1) > \overline{y}_\Gamma(\pi_c)$, $\Gamma_h(\pi_1)$ would intersect $\Gamma_v(\pi_c)$. As above, $c = 1$ is impossible.

This completes the proof if Γ_v is increasing. See Fig. 8 for an illustration of the different cases. If Γ_v is decreasing, the argument is similar but considers instead the vertices $\pi_{d-1}, \pi_d, \ldots, \pi_n$. The same geometric arguments apply. □

Combining these two claims provide the proof of Lemma 9. □

E Correctness of LsvrPaths Algorithm

Here we prove Lemma 10. We break the proof into a sequence of lemmas.

Lemma 14 *Let $A_\pi = \{1, \ldots, a\}$. If $(1, 2, \ldots, a)$ is decreasing in π, then either $A_\pi \setminus C_\pi = \emptyset$ or $A_\pi \setminus C_\pi = \{1, \ldots, a'\}$ for some $a' \leq a$.*

Proof. Let $i \in A_\pi \cap C_\pi$ (if no such i exists, we are done) and suppose that $i + 1 \in A_\pi$. Since $\pi^{-1}(i) > \pi^{-1}(i+1)$, we can write $i = \pi_{j+1}$ and $i + 1 = \pi_j$ for some j, where $\pi_{j+1} \in C_\pi$. By definition, if $\pi_{j+1} \in C_\pi$ then $\pi_j \in C_\pi$. We can now choose the minimal $i \in A_\pi \cap C_\pi$, and apply the above argument inductively to complete the proof. □

Lemma 15 *Let $A_\pi = \{1, \ldots, a\}$ and $C_\pi = \{\pi_1, \ldots, \pi_c\}$. After Step 1 there is a crossing among $\Gamma(A_\pi)$ only if $(1, 2, \ldots, a)$ is decreasing in π and among $\Gamma(C_\pi)$ only if $\pi_1 > \pi_2 > \cdots > \pi_c$.*

Proof. If $(1, 2, \ldots, a)$ is increasing in π, then after Step 1 we have $\underline{x}_\Gamma(\pi_i) < \underline{x}_\Gamma(\pi_{i+1})$ and $\underline{y}_\Gamma(i) < \underline{y}_\Gamma(i+1)$ for all $i < a$, hence there are no crossings in $\Gamma(A_\pi)$. The argument for C_π is similar. □

Lemma 16 *At the end of LsvrPaths, the required visibilities are present in Γ and no others.*

Proof. We first prove the claim for horizontal visibilities. By Lemma 14, write $A_\pi \setminus C_\pi = \{1, \ldots, a'\}$. Consider the vertices $\{a', \ldots, n\}$, whose vertical bars are not altered after Step 1. By construction, for all $i \in \{a'+1, \ldots, n\}$, $Y_\Gamma(i) = [i, i+1+\epsilon]$. Therefore, $Y_\Gamma(i) \cap Y_\Gamma(i+1) = [i+1, i+1+\epsilon]$ and $Y_\Gamma(j) \cap [i+1, i+1+\epsilon] = \emptyset$ for all $j \neq i, i+1$ (even if Step 3 is performed). Hence, $\Gamma_h(i)$ and $\Gamma_h(i+1)$ share a horizontal visibility for all $i \in \{a', \ldots, n-1\}$. Moreover, $Y_\Gamma(i) \cap Y_\Gamma(j) = \emptyset$ for $j \notin \{i-1, i, i+1\}$, so there are no unwanted visibilities among these shapes. If Step 3 is not performed, then the same argument demonstrates that all required horizontal visibilities are present among $[n]$. Suppose, therefore, that Step 3 is performed. Fix $i \in \{2, \ldots, a'\}$. To complete the proof, it suffices to show that $\Gamma_h(i)$ shares a visibility with $\Gamma_h(i-1)$ and does not share a visibility with $\Gamma_h(j)$ for $j \leq i-2$. By construction, $Y_\Gamma(i) = [2-i, i+1+\epsilon]$ for all $i \in [a']$. Hence, $Y_\Gamma(i) \supset Y_\Gamma(i-1)$. Furthermore, since Step 3 was performed, $(1, 2, \ldots, a')$ is decreasing in π

and so $\underline{x}_\Gamma(1) > \underline{x}_\Gamma(2) > \cdots > \underline{x}_\Gamma(a')$ demonstrating that $\Gamma_\mathsf{h}(j)$ for $j \leq a'$ cannot block the visibility between $\Gamma_\mathsf{h}(i)$ and $\Gamma_\mathsf{h}(i-1)$. Moreover, $\underline{y}_\Gamma(j) \geq a' + 1 \geq i + 1$ for all $j \geq a' + 1$; hence, neither can $\Gamma_\mathsf{h}(j)$ for $j \geq a' + 1$. Finally, since $Y_\Gamma(i-1) \supset Y_\Gamma(i-2) \supset \cdots \supset Y_\Gamma(1)$, we see that $Y_\Gamma(i-1)$ blocks $\Gamma_\mathsf{h}(i)$ from sharing a visibility with $\Gamma_\mathsf{h}(j)$ for $j \leq i-2$. The proof of vertical visibilities is almost identical, except that the argument uses y-projections. □

Lemma 17 *After Algorithm* LsvrPaths *is complete, there are no crossings among any shapes.*

Proof. First we observe that after Step 1 of the Algorithm, there is a crossing between two shapes $\Gamma(i_1)$ and $\Gamma(i_2)$ iff we can write $i_1 = i$ and $i_2 = i + 2$ and $(i + 1, i) \in P_\mathsf{v}$. Consequently, we may write $i + 1 = \pi_j$ and $i = \pi_{j+1}$ for some j. Since condition F1 is not met, in this case either $i \in C_\pi$ or $i+1 \in A_\pi$. First we consider the case when $i \in C_\pi$. After Step 2 is carried out, we have $\underline{x}_\Gamma(i) = 2-j-1 < 2-j = \underline{x}_\Gamma(i+1)$ and since there was no vertical displacement of the bars, this alleviates the crossing between $\Gamma_\mathsf{h}(i)$ and $\Gamma_\mathsf{v}(i-1)$. It remains to show that the transformation did not induce any further crossings. Notice that even after Step 3, the only shapes which share an x-coordinate with $\Gamma(i)$ are $\Gamma(i-1)$ and $\Gamma(i+1)$. This is because $i > a'$ where $A_\pi \setminus C_\pi = [a']$, hence the modifications to $\Gamma(A_\pi \setminus C_\pi)$ (if any) stretch the shapes *downwards* and thus $Y_\Gamma(A_\pi \setminus C_\pi) \cap Y_\Gamma(i)$ is unaffected. Furthermore, $\Gamma(i-1)$ intersects $\Gamma(i)$ iff they share a vertical visibility, in which case we must have $i - 1 = \pi_{j+2}$ (since $i + 1 = \pi_j$) and, by Lemma 16, Γ contains only the correct vertical visibilities. In this case, notice that $\pi_{j+2} \in C_\pi$ since $\pi_{j+1} \in C_\pi$ and by Lemma 15, $\pi_1 > \pi_2 > \cdots > \pi_c$ and $\pi_{j+2} = i - 1 < i = \pi_{j+1}$. This completes the proof if $i \in C_\pi$. If $i+1 \in A_\pi$ instead, the argument is similar, except we argue about Step 3 and the vertical displacement of $\Gamma(i+1)$. □

F Modifying LsvrPaths

Here we describe how to modify LsvrPaths if it is condition F2, F3, or F4 rather than F1 that is not met. Let A_π, B_π, C_π and D_π be as in Definition 1. If F2 (resp., F3, F4) is not met, we construct an LSVR Γ in which Γ_v is monotonically increasing (resp., decreasing, decreasing) and Γ_h is monotonically decreasing (resp., increasing, decreasing). Thus, for all i, we place $\Gamma(\pi_i)$ such that its corner is at $(i, n+1-\pi_i)$ (resp., $(n+1-i, \pi_i)$, $(n+1-i, n+1-\pi_i)$).

The transformation in Step 2 is either performed on $\Gamma(C_\pi)$ (as presented in LsvrPaths) or $\Gamma(D_\pi)$. If performed on $\Gamma(D_\pi)$ then $\Gamma(\pi_i)$ is stretched backwards to $i - (n-1)$ (recall that it was originally placed at x-coordinate $n + 1 - i$). Step 2 is performed on $\Gamma(C_\pi)$ only if Γ_v is increasing (cases F1 and F2). It is performed on $\Gamma(D_\pi)$ only if Γ_v is decreasing (cases F3 and F4).

The transformation in Step 3 is either performed on $\Gamma(A_\pi \setminus Z)$ (again, as presented) or $\Gamma(B_\pi \setminus Z)$, where $Z \in \{C_\pi, D_\pi\}$ is the relevant set in Step 2. If performed on $\Gamma(B_\pi)$, then $\Gamma(i)$ is stretched downwards to y-coordinate $i - (n-1)$. Step 3 is performed on $\Gamma(A_\pi)$ only if Γ_h is increasing (cases F1 and F3). It is performed on $\Gamma(B_\pi)$ only if Γ_h is decreasing (cases F2 and F4).

Importantly, the transformations of Steps 2 and 3 are only performed in certain circumstances to do with the ordering between the two paths. Specifically:

1. If Γ_v is increasing and Γ_h is decreasing, then Step 2 is performed if $\pi_1 < \pi_2 < \cdots < \pi_c$, and Step 3 if $(n, n-1, \ldots, b)$ is decreasing in π.
2. If Γ_v is decreasing and Γ_h increasing, then Step 2 is performed if $\pi_n > \pi_{n-1} > \cdots > \pi_d$ and Step 3 if $(1, 2, \ldots, a)$ is increasing in π.
3. If Γ_v and Γ_h are both decreasing then Step 2 is performed if $\pi_n < \pi_{n-1} < \cdots < \pi_d$ and Step 3 if $(n, n-1, \ldots, b)$ is increasing in π.

The proofs of correctness in these alternate cases are largely symmetric to the one presented in the case of F1.

Finding a Maximum Clique in a Grounded 1-Bend String Graph*

J. Mark Keil Debajyoti Mondal Ehsan Moradi

Abstract

A grounded 1-bend string graph is an intersection graph of a set of polygonal lines, each with one bend, such that the lines lie above a common horizontal line ℓ and have exactly one endpoint on ℓ. We show that the problem of finding a maximum clique in a grounded 1-bend string graph is APX-hard, even for strictly y-monotone strings. For general 1-bend strings, the problem remains APX-hard even if we restrict the position of the bends and end-points to lie on at most three parallel horizontal lines. We give fast algorithms to compute a maximum clique for different subclasses of grounded segment graphs, which are formed by restricting the strings to various forms of L-shapes.

1 Introduction

A *geometric intersection graph* consists of a set of geometric objects representing the nodes of the graph, where two nodes are adjacent if and only if the corresponding objects intersect. Intersection graphs that arise from the intersection of *strings*, i.e., simple curves in $\mathbb{R}^2$, are called *string graphs*. A number of restrictions on the strings have been examined in the literature. Outerstring graphs and grounded string graphs are two widely studied classes of graphs that resulted from such restrictions. An *outerstring graph* is a string graph, where the strings lie inside a disk, with one endpoint on the boundary of the disk. In a *grounded string* graph, the strings lie above a common horizontal line ℓ, with one endpoint on ℓ. The line ℓ is referred to as a *ground line*.

Although the outerstring graphs and grounded string graphs are the same for general strings, they can be different when we put restrictions on the strings. For example, if we restrict the strings to be straight line segments, the resulting grounded segment graph class is a proper subclass of outersegment graph class [3]. Strings are often modeled with polygonal chains, where a *k-bend string* is a polygonal chain with at most k bends or $(k+1)$ segments.

*Department of Computer Science, University of Saskatchewan, Saskatoon, Canada {mark.keil,dmondal,ehsan.moradi}@cs.usask.ca. The work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

While the maximum independent set problem is NP-complete for string graphs (even when the strings are straight line segments), it is polynomial-time solvable for outerstring graphs with a given outerstring representation of polynomial size [7]. Therefore, it is natural to explore other common optimization problems for outerstring graphs. In this paper we explore the *maximum clique* problem, i.e., we seek a largest subset of pairwise intersecting strings. Cabello et al. [2] proved the maximum clique problem to be NP-hard even for ray intersection graphs. Since one can enclose a ray intersection graph inside a circle such that all the rays hit the perimeter, their result implies NP-hardness for computing a maximum clique in an outersegment graph. Therefore, an interesting question that arises in this context is whether the maximum clique problem remains NP-hard for grounded segment graphs. While the problem remains open, in this paper, we show NP-hardness for two subclasses of grounded 1-bend strings.

1.1 Related Research

The hardness of independent set and maximum clique problems in general graphs inspired researchers to examine these problems for restricted intersection graph classes. Both problems remain polynomial-time solvable for *circle graphs*, i.e., the intersection of a set of chords of a circle [11, 13]. However, both become NP-complete in general segment intersection graphs [8].

A set of curves is *k-intersecting* if every pair of curves have at most k points in common. A string graph is k-intersecting if it is the intersection graph of a k-intersecting set of curves. Fox and Pach [5] gave subexponential time algorithms for computing a maximum independent set for string graphs, as well as algorithms for approximating independent set and maximum clique in k-intersecting graphs.

Middendorf and Pfeiffer [10] showed the maximum clique problem to be NP-hard for axis-aligned 1-bend strings, even when the strings are of two types: ⌝ and ⌞. They also showed the problem to be polynomial-time solvable for two cases: (a) For the strings of type ⌜ and ⌝, and (b) for grounded segments when the free endpoints of the segments lie on a fixed number of horizontal lines.

Keil et al. [7] examined the maximum independent set problem for outerstring graphs. Given an outerstring representation, where each segment is represented as a

polygonal chain, they showed how to compute a maximum independent set in $O(s^3)$ time, where s is the total number of segments in the representation. Bose et al. [1] gave an $O(n^2)$-time algorithm when the strings are y-monotone polygonal paths of constant length with segments at integral coordinates. They also showed this to be the best possible under strong exponential time hypothesis.

A rich body of research examines the recognizability of various classes of string graphs [9, 3, 12]. Throughout this paper, whenever we examine an intersection model, we assume that the input graph comes with a representation satisfying that intersection model.

1.2 Contributions

We first prove that the problem of computing a maximum clique in a grounded 1-bend string graph is APX-hard, even for strictly y-monotone strings. We then show that the problem remains NP-hard when the bend and end points of the strings (not necessarily y-monotone) are restricted to lie on three horizontal lines. Finally, we give fast polynomial-time algorithms for some restricted grounded 1-bend string graphs. In particular, when the grounded 1-bend strings are 1- and 2-sided L-shapes, and 2-sided square L-shapes. The results are summarized in the following table. Note that the class of 2-sided grounded L-shapes is known to be a proper subclass of grounded segment graph class [6]. Therefore, the time complexity question for computing a maximum clique in a grounded segment graph remains open.

Graph Class	Complexity	Reference
1-sided L-shape	$O(n^2 \log\log n)$	Sec. 5
2-sided square L	$O(n^2 \log^2 n)$	Sec. 4
2-sided L-shape	$O(n^3)$	Sec. 3
1-bend string	APX-hard	Sec. 2

Throughout the paper, we use the term *L-shape* to denote an axis-aligned 1-bend string. An L-shape intersection representation is *1-sided* (Figure 1(a)–(b)), if the L-shapes in the representation all turn clockwise, or all turn anticlockwise. Otherwise, the representation is *two-sided* (Figure 1(d)). In a *square L-shape representation*, the horizontal and vertical segments of every L-shape are of the same length (Figure 1(c)).

2 APX-hardness

In this section we show that finding a maximum clique in a grounded 1-bend string graph is an APX-hard problem, even when each string is strictly y-monotone. We reduce the maximum independent set problem in $2k$-subdivisions of cubic graphs, which is APX-hard for any

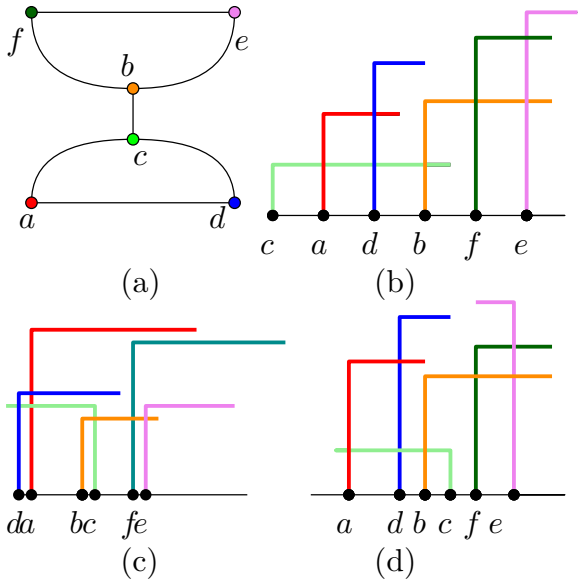


Figure 1: (a) A graph G. (b) An one-sided L-shape representation. (c) A square L-shape representation. (d) A two-sided L-shape representation.

fixed $k \geq 0$ [4]. Here a *t-subdivision* of a graph is obtained by replacing each edge (u, v) of G with a path $(u, d_1, d_2, \ldots, d_t, v)$ of t division vertices.

Let G be a 2-subdivision of a cubic graph. We first compute the complement graph $\overline{G}$ and then show how $\overline{G}$ can be represented as a strictly y-monotone grounded 1-bend string graph. Assume that there exists a $(1-\varepsilon)$-approximation algorithm $\mathcal{A}$ for the maximum clique problem in grounded 1-bend string graphs. Let C be a maximum clique obtained from $\overline{G}$ using $\mathcal{A}$. Let I^* and C^* be a maximum independent set and a maximum clique in G and $\overline{G}$, respectively. Then $|C| \geq (1-\varepsilon)|C^*|$. Note that an independent set in a graph corresponds to a clique in its complement, and vice versa. Therefore, we obtain an independent set of size at least $(1-\varepsilon)|I^*|$ in G, which contradicts the APX-hardness for computing a maximum independent set in G. Therefore, it now suffices to give an algorithm that represents $\overline{G}$ using strictly y-monotone grounded 1-bend strings.

2.1 Grounded 1-bend string representation with y-monotone strings

Assume that G is the 2-subdivision of a cubic graph H (Figure 2(a)). Let the x-axis be the ground line. While constructing the representation for $\overline{G}$, we will refer to the strings corresponding to the vertices that originally belong to H as the *original strings*, and the other strings as the *division strings*.

Note that the original vertices form a clique in $\overline{G}$. For each vertex $i = 1 \ldots, n$ in H, we construct an original string, which is a straight line segment with end-points

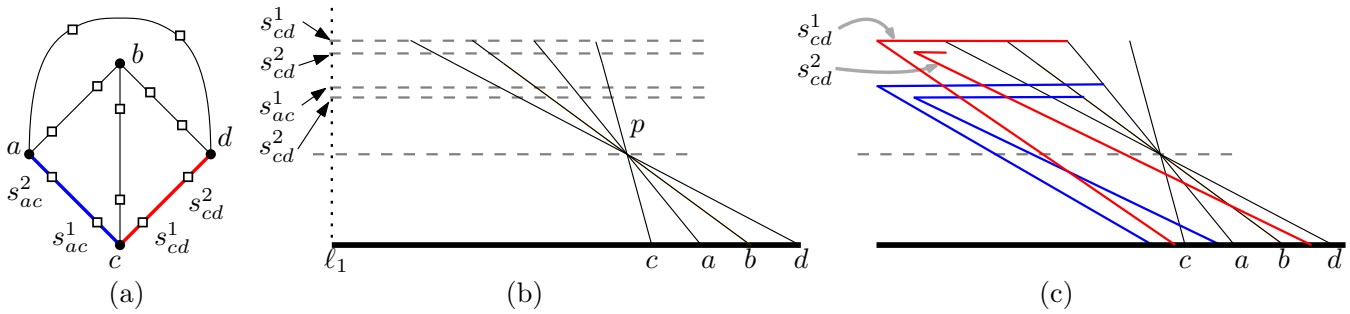


Figure 2: Illustration for the (a) graph G, and (b) the construction of the original strings. (c) Construction of a grounded 1-bend string representation for $\overline{G}$.

$(i, 0)$ and $(-3ni, 6n + 2)$. All these lines pass through the point $p = (0, 2)$. Consequently, we obtain a set of pairwise intersecting strings (Figure 2(b)).

We now describe the construction for the division strings. Let ℓ_1 be the vertical segment that starts at $(-3n^2-1, 6n+2)$ and hits the ground line (Figure 2(b)). Then there are $6n$ horizontal lines (through integral coordinates) between $y = 3$ and $y = 6n + 6$. This number is larger than $3n$, i.e., the number of division vertices in $\overline{G}$. We use these lines to create the division strings.

We number the edges of H from 1 to $(3n/2)$. Let (c, d) be the jth edge in H and let s^1_{cd}, s^2_{cd} be the corresponding division vertices in G. The vertex s^1_{cd} (resp., s^2_{cd}) is adjacent to all the original vertices and division vertices of $\overline{G}$, except for c (resp., d) and s^2_{cd} (resp., s^1_{cd}). We now construct a pair of division strings that realize these adjacencies.

Let c and d be the tth and rth vertex of H. Assume without loss of generality that the string of c appears to the left of the string of d on the ground line. The division string for s^1_{cd} starts at the ground line, makes a bend at ℓ_1, and then intersect all the original strings that lie to the left of the string for c. In particular, the string starts at $(t - \frac{1}{j}, 0)$, makes a right turn at $q = (-3n^2 - 1, 6n - j + 3)$ following the line $y = (6n - j + 3)$, and stops as soon as it intersects all the strings that appear before the string of c on this line. The division string for s^2_{cd} starts at $w = (r - \frac{1}{j}, 0)$, makes a right turn at the intersection point of the lines qw and $y = (6n - j + 2)$, and continues until it intersects all the strings that appear before the string of d (Figure 2(c)).

Since the permutation of the original strings on the ground line is opposite to the permutation on the line $y = 6n+2$, it is straightforward to verify that the string for s^1_{cd} (resp., s^2_{cd}) intersects all the original strings except the one for c (resp., d).

By construction, the strings of s^1_{cd} and s^2_{cd} are disjoint and intersect all the previously added division strings. Otherwise, suppose for a contradiction that a previously added division string z has not been intersected by the string for s^2_{cd}. Then z should appear to the left of s^2_{cd}. Thus the segment of z that touches the ground line will be almost vertical. Therefore, z will have a negative x-coordinate at the ground line, which contradicts the property that every division string starts with a positive x-coordinate. The argument is the same for s^1_{cd}.

This completes the grounded string representation for $\overline{G}$.

Since the coordinates explicitly described above are polynomial in n, the intersection of the line segments required to carry out the construction also have coordinates of polynomial size. Hence one can compute the string representation in polynomial time. Note that the strings that we computed are y-monotone. To make the strings strictly y-monotone, one can carry out the same construction for division strings with a set of slanted parallel lines of small positive slope, instead of horizontal ones. We thus have the following theorem.

Theorem 1 *The maximum clique problem is APX-hard for grounded 1-bend string graphs, even when the strings are strictly y-monotone.*

2.2 Grounded string representation on a few lines

Given a 1-bend string, we will refer to its two endpoints as *fixed* or *free* depending on whether they lie on the ground line or not. Similarly, we call its segments *fixed* or *free* depending on whether the segment is adjacent to the ground line or not.

We slightly modify the construction of the previous section such that the bends and end-points of the strings lie on at most three lines (above the ground line): ℓ_1, ℓ_2, ℓ_3, as illustrated in Figure 3. We omit the coordinate details for this construction. It is straightforward to use a very similar approach as in the previous section to compute the representation with coordinates of polynomial size.

We create the original strings such that bend-points and free endpoints lie on ℓ_2 and ℓ_1, respectively. We also ensure that the order of the free endpoints on ℓ_1 is opposite to the order of the fixed endpoints.

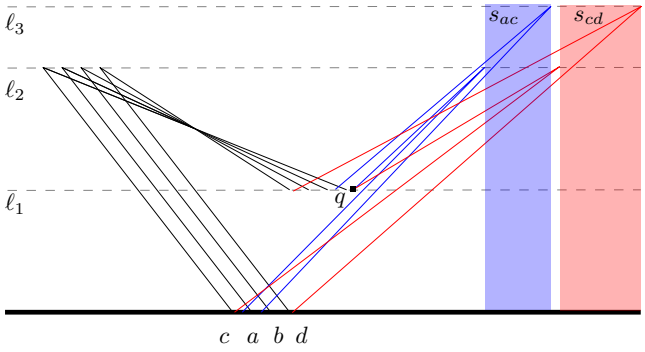


Figure 3: Construction of a grounded string representation for $\overline{G}$ on three lines.

Let q be a point to the right of all the free endpoints of the original strings on ℓ_1. A pair of division strings start at the ground line near their corresponding original strings (as in our earlier construction), but their bends are placed on ℓ_2 and ℓ_3 such that the strings remain disjoint. We also ensure that fixed segments of these strings lie to the right of q. Consider now a division string z that starts at the ground line and reaches ℓ_2 or ℓ_3. If all the required intersections are realized at its fixed segment, then we create the free segment by connecting q and its bend-point. If only a subset of the required intersections is realized at its fixed segment, then the free segment is created by connecting the bend-point to an appropriate point q' to the left of q on ℓ_1. Since the permutation of the fixed endpoints of the original strings is the reverse of the permutation of their free endpoints, such a point q' must exist.

The division vertices are created in pairs and their bend-points are placed to the right of all the previously created bend-points, as illustrated using the vertical stripes in Figure 3. Note that every newly created string z needs to reach either q or to a point to the left of q on ℓ_1. Therefore, we can choose the new bend-point of z sufficiently far apart such that its free segment crosses all the previously added division strings at their fixed segments. This completes the required construction for $\overline{G}$. We thus have the following theorem.

Theorem 2 *The maximum clique problem is APX-hard for grounded 1-bend string graphs, even when the bends and end-points are restricted to lie on three horizontal lines.*

3 Two-sided L-shapes

In this section we consider the case when the grounded strings are two-sided L-shapes. We use dynamic programming to compute a maximum clique on this class of graphs, and give an $O(n^3)$-time algorithm to compute a maximum clique. We assume that all the L-shapes are in general position, i.e., no two segments in the intersection representation lie on the same horizontal or vertical line.

Let G be an intersection graph of two-sided L-shapes, and let Q be a maximum clique with at least two vertices in G. Let a and b be the highest and second-highest L-shape in Q, respectively, and without loss of generality assume that a appears to the left of b on the ground line (Figure 4). Then any other L-shape c in Q must be below the line ℓ determined by the horizontal segment of b. Furthermore, since c intersects both a and b, its endpoint on the ground line must be to the left of a or to the right of b. In other words, the interval $[a, b]$ on the ground line acts as a forbidden interval for the other vertices in the clique.

If c is the next highest clique after b in Q, then depending on whether it lies to the left or right of the interval $[a, b]$, the forbidden region for the remaining vertices in Q grows to either $[c, b]$ or $[a, c]$. Without loss of generality assume that c lies to the right of b, and the new forbidden region is $[a, c]$. Then an L-shape intersecting c and a must also intersect b, where b already belongs to Q. One can thus continue adding a new L-shape that intersect the L-shapes representing the current forbidden interval, without worrying about the L-shapes which have been chosen already. We use this idea to design the dynamic programming algorithm.

If G does not contain any edge, then the maximum clique size is 1. Otherwise, let $D(a, b)$ denote a maximum clique, where a, b, or b, a are the first and second highest L-shapes, and a lies to the left of b. Let c be an L-shape that intersects both a and b, and for an L-shape w, let w_x be its x-coordinate on the ground line. Then

$$D(a,b) = \begin{cases} 2, \text{ if } c \text{ doesn't exist}, \\ \max \begin{cases} \max\limits_{c_x < a_x} \{D(c,b)\} + 1 \\ \max\limits_{c_x > b_x} \{D(a,c)\} + 1, \quad \text{otherwise.} \end{cases} \end{cases}$$

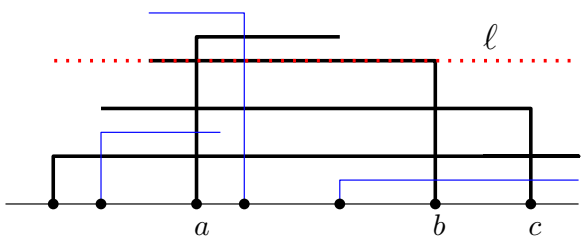


Figure 4: Illustration for the dynamic programming for two-sided L-shapes. The clique Q is shown in black.

We take the maximum over all pairs of L-shapes a, b in the input. We can use a 2-dimensional table $T(a, b)$ to store the solution of $D(a, b)$. The size of the dynamic programming table is $O(n^2)$, where n is the number of

L-shapes. Computing each entry of the table requires $O(n)$ table look-up. Therefore, it is straightforward to compute the maximum clique in $O(n^3)$ time.

The following theorem summarizes the result of this section.

Theorem 3 *Given a set of n grounded two-sided L-shapes, one can compute a maximum clique in the corresponding intersection graph in $O(n^3)$ time.*

4 Two-sided Square L-shapes

In this section, we consider two-sided square L-shapes, and give an $O(n^2 \log^2 n)$-time algorithm to compute a maximum clique. We assume the L-shapes are in general position.

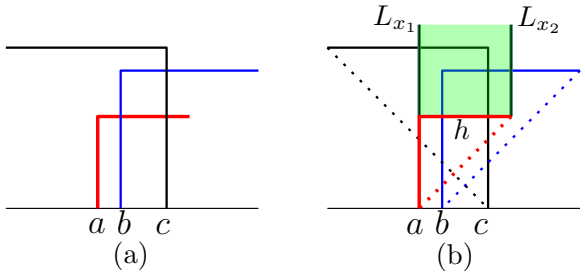


Figure 5: (a) A maximum clique in a two-sided square L-shape representation. (b) Illustration for the properties of the clique.

We first discuss some geometric properties of a maximum clique, which will help us to design the dynamic programming (Figure 5). Let a be the lowest L-shape of a maximum clique Q. Then all other L-shapes must intersect the horizontal segment h of a. Let L_{x_1} and L_{x_2} be vertical lines through the left and right endpoints of h. We now have the following observation.

Lemma 4 *Every square L-shape that intersects h, must intersect L_{x_1} or L_{x_2}.*

Proof. Suppose for a contradiction that b is an L-shape that intersects h but does not intersect L_{x_1} or L_{x_2}. Then the length of the horizontal segment of b is at most the length of h. Thus the maximum length for the vertical segment of b is also bounded by the length of h. Since the bend-point of b is above h, the vertical segment of b cannot reach the ground line, which contradicts that b is a grounded square L-shape. □

Let R be the region above h, bounded by L_{x_1} and L_{x_2}. Since every L-shape intersecting h, also intersects either L_{x_1} or L_{x_2} (Lemma 4), the parts of these L-shapes inside R corresponds to the chords of a circle graph, where the boundary of R corresponds to a part of the circle perimeter. Since a maximum clique in a circle graph can be obtained in $O(n \log^2 n)$ time [13], we can find the maximum clique containing a in the same time assuming the circle graph representation is precomputed.

Finally, we iterate the above process over all L-shapes to find the maximum clique. Let S be the input L-shapes. It is straightforward to precompute the circle graph for every L shape in $O(n^2 \log n)$ time in total. Hence the time complexity for computing maximum clique is $O(n^2 \log n) + \sum_{q \in S} O(n \log^2 n) \in O(n^2 \log^2 n)$, where S is the set of L-shapes in the input.

The following theorem summarizes the result of this section.

Theorem 5 *Given a set of n grounded two-sided square L-shapes, one can find the maximum clique of the corresponding intersection graph in $O(n^2 \log^2 n)$ time.*

5 One-sided L-shapes

In this section, we consider one-sided L-shapes, and give an $O(n^2 \log \log n)$-time algorithm to compute a maximum clique. We assume the L-shapes are in general position.

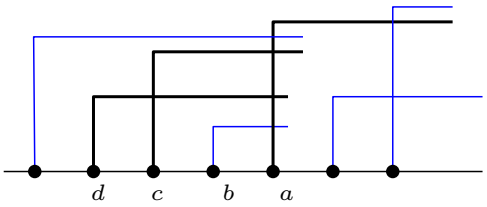


Figure 6: Illustration for the dynamic programming for one-sided L-shapes. The clique Q is shown in black.

Let S be the set of L-shapes in the input and let Q be a maximum clique. Let a be the highest L-shape in Q. Then all the other L-shapes in Q must intersect the vertical segment of a (Figure 6). There may also exist L-shapes (e.g., b) that do intersect the vertical segment of a but does not belong to Q.

Let $D(S)$ be the maximum clique in the intersection graph of the set S of input L-shapes. For any L-shape $q \in S$, let $N(q)$ be the subset of S such that every L-shape in $N(q)$ intersects the vertical segment of q. Then $D(S)$ can be defined as follows.

$$D(S) = \max_{q \in S}(D(N(q)) + 1)$$

To compute the maximum clique efficiently, we do some preprocessing. We first compute the intersection graph G of S in $O(n^2)$ time. We then compute a sorted list S_x consisting of the fixed endpoints (on the grounded line) of all the L-shapes. We next compute another sorted list S_h of the heights of the L-shapes (Figure 7). Both these lists can be computed

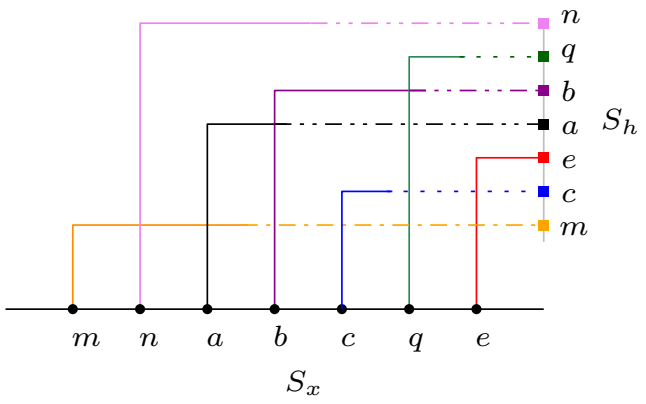


Figure 7: Illustration for S_x and S_h.

in $O(n \log n)$ time. Hence the total preprocessing takes $O(n^2) + O(n \log n) \in O(n^2)$ time.

For each $q \in S$, we now compute the maximum clique in $N(q)$. First note that $N(q)$ corresponds to a permutation graph, where the edges are determined by the intersection of the L-shapes in $N(q)$. We first find two ordered lists for the L-shapes of $N(q)$, one list corresponds to the order they appear on the ground line, and the other list corresponds to the order they appear on the vertical segment of q. We then use an $O(n \log \log n)$-time algorithm [14] for computing a maximum clique in a permutation graph to compute the maximum clique in $N(q)$.

To list the L-shapes of $N(q)$ in the order they appear on the ground line, we scan S_x from left to right and create a new ordered list S'_x with the L-shapes that intersect the vertical segment of q. We then scan S_h and create a new ordered list S'_h that contains the L-shapes intersecting the vertical segment of q. Constructing S'_x and S'_h takes $O(n)$ time. Hence computing $N(q)$ and finding a maximum clique in $N(q)$ takes $O(n) + O(n \log \log n)$ time. Thus the total running time is

$$\sum_{q \in S} O(n) + O(n \log \log n) \in O(n^2 \log \log n).$$

The following theorem summarizes the result of this section.

Theorem 6 *Given a set of n grounded one-sided square L-shapes, one can find the maximum clique of the corresponding intersection graph in $O(n^2 \log \log n)$ time.*

6 Conclusion

In this paper we have examined the maximum clique problem for the grounded 1-bend string graphs. We show the problem to be hard for y-monotone strings. We also show that the problem remains hard when we relax monotonicity, but restrict the bends and free endpoints of the strings to lie on three horizontal lines.

The most intriguing open problem is to settle the time complexity for grounded segment graphs. However, there are various questions worth investigating for 1-bend strings where the bends and endpoints lie on a few lines. If we allow only one horizontal line, then the resulting y-monotone strings become segments and the corresponding intersection graph is a permutation graph, where one can find a maximum clique in polynomial time. For a fixed number of lines the problem is polynomial-time solvable for segments [10]. Therefore, it would be interesting to examine whether the problem becomes polynomial-time solvable for two horizontal lines, where we can explore 1-bend strings. We think such restriction on the number of lines would be non-trivial even for strictly y-monotone 1-bend strings.

We have developed polynomial-time algorithms to find maximum clique for various types of L-shapes. A natural question is whether the running times of these algorithms can be improved. It would also be interesting to find non-trivial lower bounds on the time complexity.

References

[1] P. Bose, P. Carmi, M. J. Keil, A. Maheshwari, S. Mehrabi, D. Mondal, and M. Smid. Computing maximum independent set on outerstring graphs and their relatives. In *Workshop on Algorithms and Data Structures*, pages 211–224. Springer, 2019.

[2] S. Cabello, J. Cardinal, and S. Langerman. The clique problem in ray intersection graphs. In *European Symposium on Algorithms*, pages 241–252. Springer, 2012.

[3] J. Cardinal, S. Felsner, T. Miltzow, C. Tompkins, and B. Vogtenhuber. Intersection graphs of rays and grounded segments. *J. Graph Algorithms Appl.*, 22(2):273–295, 2018.

[4] M. Chlebík and J. Chlebíková. The complexity of combinatorial optimization problems on d-dimensional boxes. *SIAM J. Discret. Math.*, 21(1):158–169, 2007.

[5] J. Fox and J. Pach. Computing the independence number of intersection graphs. In D. Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1165. SIAM, 2011.

[6] V. Jelínek and M. Töpfer. On grounded l-graphs and their relatives. *Electr. J. Comb.*, 26(3):P3.17, 2019.

[7] J. M. Keil, J. S. Mitchell, D. Pradhan, and M. Vatshelle. An algorithm for the maximum weight independent set problem on outerstring graphs. *Computational Geometry*, 60:19–25, 2017.

[8] J. Kratochvíl and J. Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 31(1):85–93, 1990.

[9] J. Matousek. Intersection graphs of segments and $\exists\mathbb{R}$. *arXiv preprint arXiv:1406.2636*, 2014.

[10] M. Middendorf and F. Pfeiffer. The max clique problem in classes of string-graphs. *Discret. Math.*, 108(1-3):365–372, 1992.

[11] N. Nash and D. Gregg. An output sensitive algorithm for computing a maximum independent set of a circle graph. *Inf. Process. Lett.*, 110(16):630–634, 2010.

[12] M. Pergel and P. Rzewski. On edge intersection graphs of paths with 2 bends. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 207–219. Springer, 2016.

[13] A. Tiskin. Fast distance multiplication of unit-monge matrices. *Algorithmica*, 71(4):859–888, 2015.

[14] M. Yu, L. Tseng, and S. Chang. Sequential and parallel algorithms for the maximum-weight independent set problem on permutation graphs. *Inf. Process. Lett.*, 46(1):7–11, 1993.

Testing Balanced Splitting Cycles in Complete Triangulations*

Vincent Despré† Michaël Rao‡ Stéphan Thomassé‡

Abstract

Let T be a triangulation of an orientable surface Σ of genus g. A cycle C of T is splitting if it cuts Σ into two noncontractible parts Σ_1 and Σ_2, with respective genus $0 < g_1 \leq g_2$. The splitting cycle C is called balanced if $g_1 \geq g_2 - 1$. We define a new notion of splitting cycle approximation allowing us to show that one can rule out in polynomial time the existence of a balanced splitting cycle when the triangulation is far enough to have one. Implementing a randomized algorithm based on the same ideas, we show that large Ringel and Youngs triangulations (for instance on 22.363 vertices) have no balanced splitting cycle, the only limitation being the size of the input rather than the computation time.

1 Introduction

Let Σ be a surface and G be a graph embedded on Σ such that each face of the graph is an open disk. A splitting cycle on a surface Σ of genus at least 2 is a simple cycle (without repeated vertices) that allows to cut Σ into two parts non-homeomorphic to disks (see Figure 1). If G has genus at least 2 it may or may not have a splitting cycle and the problem is NP-complete [3, 2]. However, splitting cycles can be found certainly when G has some additional properties.

Conjecture 1 (Barnette (1982) [13, p. 166]) *If G is triangulation without loops or multiple edges of a surface of genus at least 2, then it has a splitting cycle.*

For now on, we consider only triangulations without loops or multiple edges. This conjecture is known to be true only in the case of the double torus [9]. It is formulated for triangulations but has been also investigated for G with sufficient face-width (the minimum number of faces crossed by a non-contractible curve). It is easy to build an embedded graph of face-width 2 without splitting cycles. Zha and Zhao [18] conjectured that a face-width of 3 is sufficient to obtain a splitting cycle and proved that 6 is actually enough. Triangulations

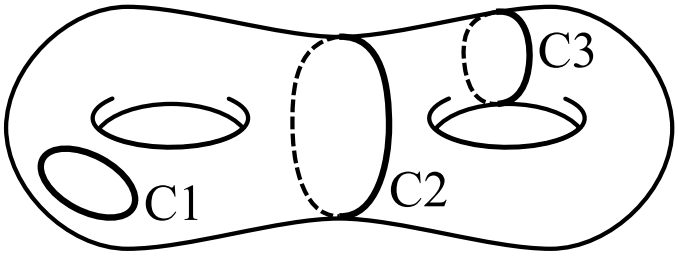


Figure 1: $C1$ is contractible, $C2$ is a splitting cycle and $C3$ is non-separating.

are a particular case of this second conjecture since any triangulation has face-width at least 3.

We say that a triangulation T is *irreducible* if none of its edges can be contracted without violating the condition of simplicity. It is easy to see that if T has a splitting cycle and is obtained by contracting an edge from some T' then T' also has a splitting cycle. Thus, it is sufficient to consider irreducible triangulations. Observe also that irreducible triangulations have face-width exactly 3. The number of irreducible triangulations of a given genus being finite [1, 14, 10], it is theoretically possible to check the conjecture for fixed genus. Sulanke gave an algorithm to compute the set of irreducible triangulations of a fixed genus [16] and used it to prove the conjecture for genus 2 with a computer assisted approach [17]. Unfortunately, the number of irreducible triangulations with respect to the genus grows too fast to hope for a brute force proof, even for genus 3.

A splitting cycle C cuts Σ into two parts of respective genus g_1, g_2, where $g_1 \leq g_2$ and $g_1 \leq g_2 = g$. We call g_1 the *type* of C, and C is called *balanced* if $g_1 \geq g_2 - 1$ (if such a cycle exists for T, we also say that T is *balanced*). It was independently conjectured by Zha and Zhao [18] and Mohar and Thomassen [13, p. 167] that a triangulation (or an embedded graph of face-width at least 3) have all the possible types of splitting cycles. However, Despré and Lazarus [4] disproved this by showing that some triangulations of complete graphs do not have all the possible types of splitting cycles. More precisely they show that some triangulations of K_{19}, the complete graph on 19 vertices or K_{43} have no balanced splitting cycle. However, the algorithm they use could not rule out the existence of balanced large complete triangulations which still could be "smoother" than small ones and allow all types of splitting cycles. The key-result of

*The first author was partially supported by the grant ANR-17-CE40-0033 of the French National Research Agency ANR (project SoS)

†Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

‡LIP, CNRS - ENS de Lyon

this paper is first to show that existence of balanced cycle in a complete triangulation T of K_n can be property tested, and then to provide an efficient implementation of this algorithm to test large complete triangulations.

Every splitting cycle C of a triangulation T partitions the edges into three classes (R, L, C), where C are the edges of the cycle, R the edges to the right of C, and L the one to the left. Moreover, in the cyclic order σ_v induced by T around the edges incident to each vertex v, the order of the types of edges is (R, C, L, C). In particular, we never have the cyclic pattern R, L, R, L. This allows a relaxation of the notion of splitting cycle. Precisely, for every $\varepsilon > 0$, an *ε-cycle* of T is a partition of the edges into three classes (R, L, U) such that:

- No vertex v have the cyclic pattern R, L, R, L in σ_v.
- All but εn of the vertices v of T are typical, i.e. every cyclic interval of σ_v of length εn contains an edge R or an edge L.

We say that an ε-cycle (R', L', U) approximates a splitting cycle (R, L, C) if $R' \subseteq R$ and $L' \subseteq L$ (here $C \subseteq U$ where U stands for unknown). Our main result is the following:

Theorem 2 *There is a randomized algorithm running in time $f(\varepsilon) \cdot poly(|T|)$ which takes as input a complete triangulation T and returns w.h.p. a set X of ε-cycles such that every splitting cycle of T is approximated by some element of X. Moreover, the size of X only depends on ε.*

Note that if T has a balanced cycle C, then the previous algorithm will find w.h.p. a balanced ε-cycle (in a sense to be defined later). Let us say that T is *ε-far* to be balanced if it does not have a balanced ε-cycle. We have the following corollary:

Corollary 3 *There is a randomized algorithm running in time $f(\varepsilon) \cdot poly(|T|)$ which takes as input a complete triangulation T which is either balanced or ε-far to be balanced and returns w.h.p. either a balanced ε-cycle, or a certificate that no balanced cycle exists.*

The previous algorithms are based on sampling a good set of vertices and can indeed be derandomized. However, even in the randomized version, the size of the family X is too large to allow any practical use. Luckily, when restricted to finding a set X approximating every balanced splitting cycle (hence cutting branches leading to unbalanced cycles), it turns out that a mix of random sampling and greedy choices can be implemented in a more efficient way. We could use this implementation in order to rule out the existence of balanced cycles in large complete triangulations. It remains to build explicit complete triangulations.

The problem of constructing triangulations of complete graphs is a very classical one, raised by Heawood in 1890 [8]. The original aim was to find an optimal proper coloring of a graph embedded on a surface of genus $g > 0$. Apart from the case of the sphere (or the plane) and the Klein bottle, the Euler formula already gives the exact upper bound of $\gamma(g) = \lfloor \frac{7+\sqrt{1+48g}}{2} \rfloor$ colors. Hence, to prove the tightness of the bound, it was necessary to produce a graph of genus g with chromatic number $\gamma(g)$. This has been achieved by Ringel and Youngs [15, 7] using complete graphs. The embeddings they provided are minimal in the sense that each complete graph cannot be embedded on a smaller genus surface and some of them are triangulations. Actually, there are many different triangulations of a given complete graph [12, 11, 6, 5]. For the experiments in this paper we will focus on the triangulations given by Ringel and Youngs for $n \equiv 7[12]$.

The major difficulty here is that the size of the sample which gives the certificate is too large to allow computation based on a one-step guess. We instead adopt a randomized greedy strategy in order to iteratively construct the sample. The algorithm is described in details in Section 4. This algorithm is extremely efficient and allow to address huge triangulations. Actually, it may be used as soon as the size of the triangulation can be stored on the computer. It has been implemented independently by Vincent Despré and Michaël Rao and they were able to reach very huge complete triangulations. Using those implementations we were able to show that the complete triangulation with 22.363 vertices (and 250.040.703 edges) given by Ringel and Youngs has no balanced splitting cycle.

Our algorithm is a new tool to deal with splitting cycles and may be useful in a larger spectrum. Indeed, when it fails to prove that the input triangulation has no balanced splitting cycles, it gives hints to find possible ones since it outputs balanced ϵ-cycles which can be the seed of some new investigation. The most appealing open question left by the paper is: Given a balanced ε-cycle, how to decide if it can be extended or not into a balanced (or near balanced) cycle. If one could design an efficient algorithm in order to find balanced splitting cycles, it would lead to efficient *divide and conquer* algorithms on complete triangulations.

We first give some properties of the splitting cycles in Section 2. Then, we prove Theorem 2 in Section 3. Section 4 is devoted to the description of the practical algorithm and the implementations details along with the different results are developed in Section 5.

2 Properties of Splitting Cycles of Complete Triangulations

We begin by fixing some notations. Let T_n be a triangulation of the complete graph K_n. We denote by $(v_0, \dots, v_n - 1)$ the vertices of K_n. Around the vertex v_i, the ordering of its neighbors is a permutation σ_i of $\{0, \dots, n-1\} \setminus i$. We will need to split the neighborhood of the vertices into parts as follows: if v_i is a vertex we denote by $(ev_0, ev_1, ev_2)_i$ a partition of the vertices around v_i such that each set ev_i contains consecutive vertices with respect to σ_i. We call a *local configuration* a couple $(i, c)_v$ where i corresponds to the part ev_i and c is a color and a *configuration* a list of local configurations.

Lemma 4 *Let v_i be a vertex of T_n, $(ev_0, ev_1, ev_2)_i$ be any partition of the edges in the neighborhood of v_i and (R, L, C) be a splitting cycle of T_n. At least one of the ev_j is entirely colored L or R.*

Proof. C may reach at most two of ev_0, ev_1 and ev_2 since it is a simple cycle. It implies that one of the ev_i has to be colored entirely L or R for any splitting cycle. □

The following lemma is a direct consequence of the previous one.

Lemma 5 *There is at least one configuration $((i_0, c_0)_{v_0}, \cdots, (i_{k-1}, c_{k-1})_{v_{k-1}})$ realized by each splitting cycle (R, L, C).*

Let us now consider the particular properties of balanced splitting cycles of complete triangulations. Indeed, as we will prove in the next lemma, a balanced splitting cycle cannot be too short in a complete graph because of the Euler characteristic $\chi(T_n) = n - e + f = 2 - 2g$ where e is the number of edges of T_n and f its number of triangles.

Lemma 6 *Let $\mathcal{C} = (R, L, C)$ be a balanced splitting cycle of T_n. Then,*

$$|C| \geq \left\lceil \frac{5 + \sqrt{2n^2 - 14n + 25}}{2} \right\rceil$$

$$\min(|L|, |R|) \geq \left\lceil \frac{n^2 - 7n + 8 + 4\sqrt{2n^2 - 14n + 25}}{4} \right\rceil$$

Proof. Since we consider complete graphs, it is not possible that two vertices be colored entirely R for one and entirely L for the other one. Hence, after cutting along C, there is a graph embedding with one boundary and no interior vertex of genus at least $\lfloor \frac{g}{2} \rfloor$. Let $k = |C|$ and T' be the triangulation without interior vertices obtained after cutting along C. T' has genus at least $\lfloor \frac{g}{2} \rfloor$ and so $\chi(T') \leq 2 - 2\lfloor \frac{g}{2} \rfloor - 1 \leq 2 - (g-1) - 1 = 2 - g$. M' has k vertices, $e \leq \frac{k(k-1)}{2}$ edges and f faces. The double counting of the number of edges gives $3f = 2e - k$ because all the edges are on exactly 2 faces except the k on the boundary. So $\chi(T') = k - e + 2\frac{e}{3} - \frac{k}{3} = \frac{2k-e}{3} \geq \frac{4k-k(k-1)}{6} = \frac{5k-k^2}{6}$. By putting together the two inequalities we obtain: $2 - g \geq \frac{5k-k^2}{6}$ leading to $k^2 - 5k + 6 - 6g \geq 0$. $\Delta = 25 - 4(6 - 6g) = 1 + 24g$ and so $k = |C| \geq \frac{5+\sqrt{1+24g}}{2} = \frac{5+\sqrt{1+2(n-3)(n-4)}}{2} = \frac{5+\sqrt{2n^2-14n+25}}{2}$.

Let us look back at the Euler formula for T'. We have, $\chi(T') = \frac{2k-e}{3} \leq 2 - g$. It implies that $e \geq 2k + 3g - 6 \geq 5 + \sqrt{2n^2 - 14n + 25} + \frac{3(n-3)(n-4)}{12} - 6 = \frac{(n-3)(n-4)+4\sqrt{2n^2-14n+25}-4}{4} = \frac{n^2-7n+8+4\sqrt{2n^2-14n+25}}{4}$. □

It is interesting to notice that $\frac{e}{\min(|L|,|R|)} = \frac{1}{2} - O(\frac{1}{n})$ for balanced splitting cycles in complete triangulations and thus $minE(n) = \frac{n^2}{4} - O(n)$.

3 Approximations of splitting cycles

Our goal is to prove Theorem 2, which shows that we can efficiently find a set X of ε-cycles approximating all splitting cycles.

Lemma 7 *For every orientable triangulation T_n of K_n and every $\varepsilon > 0$, there is a set X of size $f(\varepsilon)$ consisting of ε-cycles such that every splitting cycle of T_n is approximated by some element of X.*

Proof. Pick some large constant $c > 4/\varepsilon^2$. We implicitly assume here that n is much larger than ε and c, otherwise X simply exists by enumeration. Pick uniformly at random a sample S of c different vertices of T_n. For each $v_i \in S$, divide the cyclic order σ_i into c cyclic intervals $I_1, \dots, I_c$ of approximately the same length (i.e. size $\lfloor (n-1)/c \rfloor$ or $\lceil (n-1)/c \rceil$). We now construct our ε-cycles (R, L, U). We first decide for each $v_i \in S$ an R, L, U (right, left, unknown) coloring of the intervals I_j in such a way that two (possibly identical) intervals are U and these two U intervals separates the R intervals and the L intervals. Note that when the U intervals are identical or adjacent, the remaining intervals are all colored R or all colored L. The total number of such choices for a given $v_i \in S$ is $c^2 + c$, and we then have $(c^2 + c)^c$ possible ways of coloring the edges adjacent to S according to this local rule. Among these coloring, some of then are inconsistent in the sense that they give both colors R and L at the two endpoints of some edge between two elements of S. We reject these colorings. It can also happen that an edge receives both colors U and R (or U and L) in which case the edge keeps the color different from U. We then color U all edges which

were not incident to vertices of S. We reject all colorings which contain the forbidden pattern (R, L, R, L) in some σ_i. The set of surviving (R, L, U) colorings is denoted by X_S, and this is our candidate for X. Note that the size of X_S only depends on c and hence on ε, and that the total number of U edges incident to vertices of S is at most $c \cdot 2n/c$.

The key-observation is that every splitting cycle C of T_n is approximated by some element of X_S. Indeed, for each vertex $v_i \in S$ one can define the two U intervals of σ_i as these containing an edge of C, and the R and L intervals are the one which are entirely R or L according to cycle C. So to reach our conclusion, we just have to show that every element of X_S is an ε-cycle.

We claim that this happens if we are lucky enough with our sampling S. Let us say that a vertex v_i is *good* if S is well distributed in σ_i i.e. if for every cyclic interval of σ_i of size at least εn, the number of elements of S is at least $\varepsilon c/2$. Observe that the probability that a vertex is good tends to 1, when ε is fixed and c goes to infinity. By Markov, we can fix c large enough such that with high probability, our sampling S will be such that all vertices save an arbitrarily small proportion are good. We now claim that in this case, all (R, L, U) partitions of X_S are ε-cycles.

Assume for contradiction that this is not the case. Then there are more than εn non typical vertices v_i for which σ_i contains an interval Ij_i of size at least εn with no $R \cup L$ edge. Since we can neglect these vertices v_i which are either in S or are not good, each of these intervals Ij_i contains $\varepsilon c/2$ vertices of S, and none of them have created an $R \cup L$ edge with v_i. So the total number of U edges incident to vertices of S is at least $\varepsilon n.\varepsilon c/2$, which is contradicting the fact that there are at most $c.2n/c$ of them since $c > 4/\varepsilon^2$. □

This concludes the proof of Theorem 2, the algorithm simply returning X_S for some large enough sample S. The main drawback of this approach is the size of the sampling, which makes it very difficult to implement for some practicle use. Since our goal is to look for balanced splitting cycles, we will only focus on ε-cycles which can be approximations of balanced cycles. Let us denote by $minE(n)$ the minimum size of R (or equivalently of L) in a balanced cycle (R, L, C) of an orientable triangulation of K_n. Note that $minE(n) = n^2/4 - O(n)$, but a more precise value will be given later when we will discuss the implementation. Thus if some ε-cycle (R', L', U) approximates (R, L, C), it must have potentially at least $minE(n)$ many R' or L' edges. Let us properly define this. The *right-potential* $r(v_i)$ of some vertex v_i is defined as:

- When v_i is incident to some edges of R' and L', $r(v_i)$ is the size of the longest cyclic interval of σ_i with a point in R' and no point in L', minus 2.
- When v_i is only incident to edges of R', we have $r(v_i) = n - 1$.
- When v_i is only incident to edges of L', $r(v_i)$ is the size of the longest cyclic interval of σ_i with no point in L', minus 2.

The same definition applies for left potential $l(v_i)$. The *right-potential* $r(R', L', U)$ is the sum of the right potential of all the vertices (same for left-potential $l(R', L', U)$). Note that $r(R', L', U) \geq 2|R|$ and $l(R', L', U) \geq 2|L|$ when (R', L', U) approximates (R, L, C) (the factor 2 in the inequality stands for the fact that we are doubly counting edges in the potential). Let us then say that an ε-cycle (R', L', U) is *unbalanced* if $r(R', L', U) < 2minE(n)$ or $l(R', L', U) < 2minE(n)$ (otherwise it is *balanced*). A triangulation T_n is *ε-far to be balanced* if it has no balanced ε-cycle.

Proof. [Proof of Corollary 3] Now let us prove that we can efficiently separate triangulations which are either balanced or ε-far to be balanced. For this, we compute a set X_S of ε-cycles which approximates all splitting cycles of T_n. Note that if T_n admits a balanced cycle (R, L, C), then it is approximated by some ε-cycle (R', L', U) in X_S which hence must be balanced and thus a certificate of separation. Now if T_n does not admit a balanced cycle (R, L, C), we compute a set X_S coming w.h.p. from a lucky sample S. The key point is that we can indeed check if S is a good sample or not, just by checking if it is well-distributed in nearly all σ_i. Hence the set X_S probably approximate all splitting cycles of T_n, and if we satisfy the separation hypothesis of Theorem 3, none of the ε-cycles are balanced. Therefore X_S is a certificate of the fact that T_n has no balanced splitting cycle. □

The nice feature of this property testing algorithm is that if we try to check if a given T_n has a balanced cycle, we may be lucky and get a NO-certificate. This is basically what happens so far for all Ringel and Youngs triangulations on which the algorithm terminates. However, in the present form, the size of X_S is way too large to be implemented, and we will use a mix of random sampling and greedy choices for S. Also the fact that we divide σ_i into c intervals is convenient for the proof but not for the algorithm, which will only cut into 3 parts.

Another exciting direction of research is when we get a set X_S of ε-cycles, some of which are balanced. There is possibly a way to investigate if a given balanced ε-cycle can be completed into a balanced (or near balanced) cycle. For instance, if some σ_i contains the pattern (R, U, R, L), then the U edge can be turned into an R edge (possibly creating forbidden patterns leading to reduction of X_S). These closure operations (together

with a (L, U, L, R) rule) can greatly densify our candidate ε-cycle making it easier to complete or not into a splitting cycle.

4 Practicle algorithm

Sketch

We choose at random a set of k vertices $(v_0, \cdots, v_{k-1})$ of T_n and $(ev_0, ev_1, ev_2)_j$ a balanced partition of the edges around v_j, for all $0 \leq j < k$. We have $3 \cdot 6^{k-1}$ different configurations on the chosen vertices since each vertex has 3 possible ev and 2 possible colors for each case. We say that a configuration is valid if it is compatible with the existence of a balance splitting cycle. We want to show that no configuration is valid and thus conclude that no balanced splitting cycle exists. By considering the other vertices of the graph, we obtain two tools to show that the configuration is not valid:

- There is a vertex with an alternated sequence of edges labeled (L, R, L, R).
- The biggest number of edges colored R (or L) that the graph can admit is less than $minE(n)$ (see Lemma 6).

It is natural to see the $3 \cdot 6^{k-1}$ as the leaves of a tree where each layer i adds the local configuration of v_i. We can remark that, if a partial configuration on a node is already invalid then all the corresponding subtree is invalid. It implies a natural breadth is the tree of configuration considered as a search tree.

Data structure

To be able to correctly describe our algorithm and analyze its complexity, it is necessary to describe a bit the data structure we use. It is mainly a half-edge data structure which consists in coding T_n by a set of half-edges each having a handle to the opposite half-edge (represented by an involution α_0) and to the next half-edge in the local permutation σ_i (we can think of it as a global permutation σ whose cycles are the σ_i). At this point, we can notice that the size of the map is actually $2e \cdot <$ size of an half-edge $>= O(e)$. An edge is an orbit of the action of α_0 on the set of half-edges and can be stored as one element in the orbit. Similarly, the orbits of σ are the vertices, it is again sufficient to store one half-edge for each vertex. We need to store on each vertex a "reverse" dictionary Rev_i that associate to every vertex v_j for $j \neq i$ its position around v_i (each vertex is associated to a unique half-edge around v_i). The Rev dictionaries are not a general feature in the half-edge data structure but is required by our algorithm. Finally, the faces can be construct by alternatively applying α_0 and σ and storing a half-edge for each corresponding orbit. Here, computing the faces is mainly useful to check that T_n is a correct triangulation. The construction of the map is considered as a precomputation and is done using $O(e)$ operations.

Algorithm

INPUT: A complete triangulation.

- Let C be an empty vector of configurations. We initialize $RandV$ with a random vertex v_i and a random partition of the neighborhood of v_i into three consecutive parts $(ev_0, ev_1, ev_2)_i$. We put the configuration $(v_i, (ev_0, ev_1, ev_2)_i, 0, L)$ in C.
- We add a list L_j on each vector v_j that stores the position of the vertices already colored. At this stage, it means that for all $v_j \in ev_0$ we call $Rev_j(i)$ to know the position of v_i around v_j and we put $(Rev_j(i), L)$ in L_j. Notice that the L_is must be sorted during the algorithm.
- While C is non-empty we do:
  1. We test if C is valid. This implies two tests:
     - We look at all the L_i to see if there is no cyclic subsequence of the form (L, R, L, R).
     - We sum the biggest interval that can be colored L (resp. R) in all the L_is and we compare the result to the one of Lemma 6.
  2. If one of the test fails we update C in the following way:
     - If the last element of C is of the form $(\cdots, 2, R)$ then we discard it and we update C again.
     - Else we consider the next configuration using the order: $(0, L)$,$(1, L)$,$(2, L)$,$(0, R)$,$(1, R)$ and $(2, R)$.

     We update each L_i to make it coherent with the new configuration and the go back to step 1.
  3. We compute a new random vertex v_i not already used by C with a partition of its neighborhood and we add $(v_i, (ev_0, ev_1, ev_2)_i, 0, L)$ at the end of C. We then update the L_i and go back to 1.

Analysis of the algorithm

Proposition 8 *If the algorithm terminates then the input triangulation does not have a balanced splitting cycle.*

Proof. If the algorithm terminates then C has described a search tree $\mathcal{T}$ rooted at the empty configuration. All the leaves of $\mathcal{T}$ corresponds to configurations that are invalid in step 1. Now, if all the children of a given node are invalid, it implies that the configuration of the node is invalid. So, by induction, all configurations in $\mathcal{T}$ are invalid and this includes its root. If the empty set is invalid, it implies that the input triangulation has no balanced splitting cycle. □

Proposition 9 *The algorithm described above requires $O(t \cdot d \cdot n) = O(t \cdot d \cdot \sqrt{e})$ operations where t is the size of the search tree $\mathcal{T}$ and d its depth.*

Proof. Each node of $\mathcal{T}$ corresponds to one iteration of the while loop. Step 1 requires reading all the lists L_i. There are n such lists and their size is bounded by the size of C which is less than the depth of $\mathcal{T}$. It implies that this step requires $O(d \cdot n)$ operations. Step 2 and 3 may require an insertion or a deletion in one third of the L_i which is done in $O(d \cdot n)$ operations. Since we consider t configurations, we obtain a total of $O(t \cdot d \cdot n)$ operations. □

5 Implementation details and experimental results

The algorithm can be made parallel by having a master thread assigning different subtrees of the search tree to different threads. There are no difficulty here and no significant risk of bug since each thread has its own copy of the data structure. The implementation has been realized in C++ using OPENMPI for parallelization and can be downloaded at `http://vdespre.free.fr/Splitting.tar.gz`. The tests have been launched on the cluster Grid'5000[1]. We denote by m be the number of threads for given experiment.

We first give results to show the efficiency of the algorithm. Notice that the limit is set by the RAM on each node and so the number of threads is set to not break the memory limit. The time column shows the average on 10 tries.

s	n	e	time (s.)
833	10 003	50 025 003	425
1863	22 363	250 040 703	2990

m	nodes	t	CPU time
180	45	2 000 000	21h15m
45	45	1 700 000	37h22m

It is interesting to notice that the time of the tests highly depends on the exact value of n. It means that the size of the research tree is not smooth with respect to n. It is pretty surprising and we have no hint of the reason by now. The following experiments have been done using 720 threads on 45 nodes.

s	n	time (s.)	*s.d.* (s.)	d_0	t
100	1207	18	1	7	1 800 000
101	1219	62	15	7	2 100 000
102	1231	945	224	9	41 000 000
103	1243	970	178	9	42 000 000
104	1255	17	1	7	1 800 000
105	1267	fails (7200)		10	
106	1279	35	8	7	1 900 000
107	1291	42	4	7	1 900 000
108	1303	220	45	7	8 200 000
109	1315	17	1	7	1 800 000
110	1327	18	1	7	1 800 000

6 Conclusion

The structure of the splitting cycles in triangulations of complete graphs remains quite mysterious. Even for the case of Ringel and Youngs embeddings restricted to $n = 12s + 7$, we do not understand what exactly happens. Our new experimental results give some information on the absence of balanced splittings. In this specific case, we can imagine to make tests on bigger triangulations by storing the embedding using $O(n)$ memory. This can be done using the extreme symmetry of the embeddings but is not likely to be generalized.

We can also want to explore other triangulations of complete graphs. A very simple question remains open on this subject:

Question 10 *Is there an unbounded sequence of triangulations of complete graphs admitting balanced splitting cycles?*

The question is of intrinsic interest and it is difficult to have an intuition about it. The constructions of triangulations of complete graphs are pretty intricate and it is not clear if one can be modified to ensure the existence of a balanced splitting. In addition, we always look for an easy proof that some triangulation does not have a splitting cycle. We think that Lemma 7 is the kind of idea that can lead to such a proof. However, it is not clear how much the properties of a specific embedding must be used. If there exists huge triangulations of complete graphs with balanced splittings, it would be necessary to use an explicit embedding. If not, we can imagine proving the non-existence of balanced splitting in complete triangulations without considering a specific embedding which is very convenient, in particular for probabilistic arguments.

[1]Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

References

[1] D. W. Barnette and A. L. Edelson. All orientable 2-manifolds have finitely many minimal triangulations. *Israel Journal of Mathematics*, 62(1):90–98, 1988.

[2] S. Cabello, É. Colin de Verdière, and F. Lazarus. Finding cycles with topological properties in embedded graphs. *SIAM J. Discrete Math.*, 25(4):1600–1614, 2011.

[3] E. W. Chambers, É. C. De Verdière, J. Erickson, F. Lazarus, and K. Whittlesey. Splitting (complicated) surfaces is hard. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 421–429. ACM, 2006.

[4] V. Despré and F. Lazarus. Some triangulated surfaces without balanced splitting. *Graphs and Combinatorics*, 32(6):2339–2353, 2016.

[5] M. N. Ellingham and C. Stephens. Triangular embeddings of complete graphs (neighborly maps) with 12 and 13 vertices. *Journal of Combinatorial Designs*, 13(5):336–344, 2005.

[6] M. J. Grannell and M. Knor. A lower bound for the number of orientable triangular embeddings of some complete graphs. *Journal of Combinatorial Theory, Series B*, 100(2):216–225, 2010.

[7] J. L. Gross and T. W. Tucker. *Topological graph theory*. Dover, reprint 2001 from Wiley edition, 1987.

[8] P. Heawood. Map-color theorem. *Quart. J. Math. Oxford Ser. 24*, 1890.

[9] D. L. Jennings. *Separating Cycles in Triangulations of the Double Torus*. PhD thesis, Vanderbilt University, 2003.

[10] G. Joret and D. R. Wood. Irreducible triangulations are small. *Journal of Combinatorial Theory, Series B*, 100(5):446–455, 2010.

[11] V. P. Korzhik and H.-J. Voss. On the number of nonisomorphic orientable regular embeddings of complete graphs. *Journal of Combinatorial Theory, Series B*, 81(1):58–76, 2001.

[12] S. Lawrencenko, S. Negami, and A. T. White. Three nonisomorphic triangulations of an orientable surface with the same complete graph. *Discrete Mathematics*, 135(1):367–369, 1994.

[13] B. Mohar and C. Thomassen. *Graphs on Surfaces*. Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001.

[14] A. Nakamoto and K. Ota. Note on irreducible triangulations of surfaces. *Journal of Graph Theory*, 20(2):227–233, 1995.

[15] G. Ringel. *Map color theorem*, volume 209. Springer, 1974.

[16] T. Sulanke. Generating irreducible triangulations of surfaces. arXiv:math/0606687, 2006.

[17] T. Sulanke. Irreducible triangulations of low genus surfaces. arXiv:math/0606690, 2006.

[18] X. Zha and Y. Zhao. On non-null separating circuits in embedded graphs. *Contemporary Mathematics*, 147:349–349, 1993.

A Linear-Time Algorithm for Discrete Radius Optimally Augmenting Paths in a Metric Space

Haitao Wang * Yiming Zhao †

Abstract

Let P be a path graph of n vertices embedded in a metric space. We consider the problem of adding a new edge to P so that the radius of the resulting graph is minimized, where any center is constrained to be one of the vertices of P. Previously, the "continuous" version of the problem where a center may be a point in the interior of an edge of the graph was studied and a linear-time algorithm was known. Our "discrete" version of the problem has not been studied before. We present a linear-time algorithm for the problem.

1 Introduction

Let P be a path graph of n vertices embedded in a metric space. We wish to add a new edge to P so that the radius of the resulting graph is minimized, where any center of the graph is constrained to be one of the vertices of P. The problem is formally defined as follows.

Let $\{v_1, v_2, ..., v_n\}$ be the set of vertices of P along P. For each $i \in [1, n-1]$, let $e(v_i, v_{i+1})$ denote the edge connecting v_i and v_{i+1}. We assume that P is embedded in a metric space and $|v_iv_j|$ is the distance between two vertices v_i and v_j, such that the following properties hold: (1) $|v_iv_j| = 0$ if and only if $i = j$; (2) $|v_iv_j| = |v_jv_i| \geq 0$; (3) $|v_iv_k| + |v_kv_j| \geq |v_iv_j|$ for any v_k (i.e., the triangle inequality). Note that the length of each edge $e(v_i, v_{i+1})$ for $i \in [1, n-1]$ in P is equal to $|v_iv_{i+1}|$. We assume that the distance $|v_iv_j|$ can be obtained in $O(1)$ time for any two vertices v_i and v_j of P.

Let $P \cup \{e(v_i, v_j)\}$ denote the resulting graph (also called *augmenting path*) after adding a new edge $e(v_i, v_j)$ connecting two vertices (i.e., v_i and v_j) of P. A vertex c of P is called a *center* of the new graph $P \cup \{e(v_i, v_j)\}$ if it minimizes the maximum length of the shortest paths from c to all vertices in the graph, and the maximum shortest path length is called the *radius* of $P \cup \{e(v_i, v_j)\}$. The problem is to add a new edge $e(v_i, v_j)$ such that the radius of $P \cup \{e(v_i, v_j)\}$ is minimized, among all vertex pairs (v_i, v_j) with $1 \leq i < j \leq n$. We call it *discrete radius optimally augmenting path* problem (or discrete-ROAP for short).

To the best of our knowledge, the problem has not been studied before in the literature. In this paper, we present an $O(n)$ time algorithm for the problem.

1.1 Related work

Johnson and Wang [12] studied a "continuous" version of the problem in which a center may be in the interior of an edge of the graph. In contrast, in our problem any center has to be a vertex of the graph, and thus our problem may be considered a "discrete" version. Johnson and Wang [12] gave a linear time algorithm for their continuous problem.

A similar problem that is to minimize the diameter of the augmenting path has also been studied. Große et al. [9] first gave an $O(n \log^3 n)$ time algorithm; later Wang [16] improved the algorithm to $O(n \log n)$ time. Variations of the diameter problem (i.e., add a new edge to P to minimize the diameter of the resulting graph) were also considered. If the path P is embedded in the Euclidean space $\mathbb{R}^d$ for a given constant d, Große et al. [9] proposed an algorithm that can compute a $(1+\epsilon)$-approximate solution for the diameter problem in $O(n + \frac{1}{\epsilon^3})$ time, for any $\epsilon > 0$. If P is embedded in the Euclidean plane $\mathbb{R}^2$, De Carufel et al. [5] derived a linear-time algorithm for the continuous version of the diameter problem (i.e., the diameter is defined with respect to all points of the graph, including the points in the interior of the graph edges, not just vertices). For a geometric tree T of n vertices embedded in the Euclidean plane $\mathbb{R}^2$, De Carufel et al. [6] designed an $O(n \log n)$-time algorithm for adding a new edge to T to minimize the continuous diameter in the new graph. If T is a tree embedded in a metric space, Große et al. [10] solved the discrete diameter problem in $O(n^2 \log n)$ time; Bilò [3] improved the algorithm to $O(n \log n)$ time. Oh and Ahn [14] considered the diameter problem on a general tree (not necessarily embedded in a metric space) and developed $O(n^2 \log^3 n)$ time algorithms for both the discrete and the continuous versions of the diameter problem; later Bilò [3] gave an improved $O(n^2)$ time algorithm for the discrete diameter problem.

A more general problem is to add k edges to a general graph G such that the diameter of the new graph

*Department of Computer Science, Utah State University, Logan, UT 84322, USA. haitao.wang@usu.edu

†Corresponding author. Department of Computer Science, Utah State University, Logan, UT 84322, USA. yiming.zhao@aggiemail.usu.edu

is minimized. This problem is NP-hard [15] and some variants are even W[2]-hard [7, 8]. Various approximation algorithms are known [4, 7, 13]. The problem of bounding the diameters of the augmenting graphs have also been studied [1, 11]. In a geometric setting, given a circle in the plane, Bae et al. [2] considered the problem of inserting k shortcuts (i.e., chords) to the circle to minimize the diameter of the resulting graph.

As a motivation of our problem, we borrow an example from [12]. Suppose there is a highway that connects several cities and we want to build a facility along the highway to provide certain service for all these cities; it is required that the facility be located in one of the cities along the highway. To reduce the transportation time, one option is to construct a new highway connecting two cities such that the radius (the maximum distance from the facility to all cities) is as small as possible.

1.2 Our approach

Note that the radius of $P \cup \{e(v_i, v_j)\}$ may not be equal to its diameter divided by 2. For example, suppose $P \cup \{e(v_i, v_j)\}$ is a cycle (i.e., $i = 1$ and $j = n$) and all edges of the cycle have the same length; then one can verify that the radius of the graph is equal to its diameter.

To solve our problem, a natural idea is to see whether the algorithm [12] for the continuous problem can be used. To this end, two basic questions arise. First, for an augmenting graph $P \cup \{e(v_i, v_j)\}$, how far a continuous center can be from the discrete center? For example, is it the case that if a continuous center lies in the interior of an edge e, then one of the two vertices of e must be a discrete center? Second, is it the case that an optimal solution (i.e., the new edge to be added) in the continuous version must also be an optimal solution for the discrete version?

In order to answer these questions, we illustrate two examples.

Figure 1 shows an example in which the path P with 10 vertices is embedded in the Euclidean plane, with $|v_iv_{i+1}| = 1$ for all $1 \leq i \leq 9$. Suppose a new edge $e(v_3, v_8)$ is added. It is possible to draw the figure such that $|v_3v_8| = 4$. One can verify that the only continuous center is the middle point of $e(v_3, v_8)$ (whose farthest vertices are $\{v_1, v_5, v_6, v_{10}\}$) and the continuous radius is 4. Either v_5 or v_6 can be a discrete center (v_5 has only one farthest vertex v_{10} and v_6 has only one farthest vertex v_1) and the discrete radius is 5. This example shows that the discrete center and the continuous center could be "far from" each other. Therefore, it is not obvious to us whether/how a continuous center can be used to find a discrete center.

Figure 2 shows an example in which the path P with 10 vertices is embedded in the Euclidean plane, with $|v_iv_{i+1}| = 1$ for all $1 \leq i \leq 9$. It is possible to draw the figure such that $|v_3v_8| = 4$, $|v_4v_7| > 2$, $|v_5v_{10}| > |v_1v_6|$,

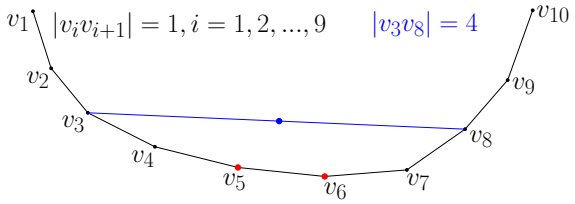


Figure 1: Illustrating the difference between the continuous center and the discrete center. The continuous center is the middle point of the new edge $e(v_3, v_8)$. Either v_5 or v_6 can be a discrete center.

and $4 < |v_1v_6| < 5$. For the continuous problem, an optimal solution is to add the edge $e(v_3, v_8)$, after which the continuous center of the new graph is the middle point of $e(v_3, v_8)$ (which has four farthest vertices $\{v_1, v_5, v_6, v_{10}\}$) and the continuous radius is 4. For the discrete problem, an optimal solution is to add the edge $e(v_1, v_6)$, after which the discrete center of the new graph is v_6 (which has only one farthest vertex v_1) and the discrete radius is equal to $|e(v_1, v_6)|$, which is larger than 4. Note that $e(v_5, v_{10})$ is not an optimal solution due to $|v_5v_{10}| > |v_1v_6|$. This example shows that optimal solutions of the two versions of the problem could be very different. Therefore, it is not obvious to us whether/how a continuous optimal solution can be used to find a discrete optimal solution.

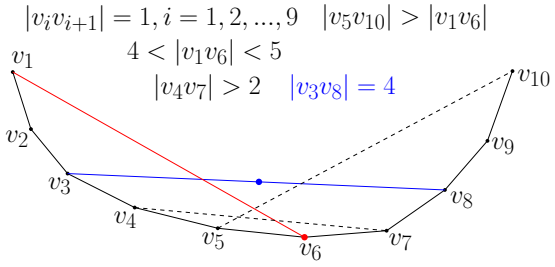


Figure 2: Illustrating the difference between an optimal solution of the continuous problem and that of the discrete problem. For the continuous problem, an optimal solution is to add the edge $e(v_3, v_8)$. For the discrete problem, an optimal solution is to add $e(v_1, v_6)$.

The above examples demonstrate that using the algorithm in [12] directly to solve the discrete problem seems not possible. Instead, we design a new algorithm. Our algorithm still share some similarities with that in [12] in the following sense. In the continuous case, any center must have two different farthest vertices in the augmenting graph. Based on the location of the center, the locations of the two farthest vertices, and the shortest paths from the center to the two farthest vertices in an optimal solution, the algorithm in [12] considers a constant number of configurations, and in each configuration the algorithm computes a candidate solution such

that if an optimal solution conforms to the configuration, then the candidate solution is an optimal solution. In our discrete case, we also consider a constant number of configurations and process the configurations in the same way as above. However, the major difference is that the definitions of the configurations in our problem are quite different from those in [12]. Indeed, in our problem, a center may have only one farthest vertex. Therefore, the configurations in our problem are defined with respect to the locations of the center and a single farthest vertex, as well as their shortest path. In addition, unlike those in [12], we do not need to consider the configurations where the center is in the interior of the new added edge. For this reason, we have much fewer configurations than those in [12].

In the following, Section 2 introduces notation and definitions. Our algorithm is described in Section 3.

2 Preliminaries

Unless otherwise stated, for any index pair (i, j) or vertex pair (v_i, v_j) used in our discussion, we assume that $1 \leq i \leq j \leq n$. For any two vertices v_i and v_j of the path P, we use $P(v_i, v_j)$ to refer to the subpath of P from v_i to v_j inclusively.

Define $G(i, j) = P \cup \{e(v_i, v_j)\}$, i.e., the new graph after a new edge $e(v_i, v_j)$ is added to the path P. Note that if $j = i$ or $j = i + 1$, then $G(i, j)$ is P. Define $C(i, j) = P(v_i, v_j) \cup e(v_i, v_j)$, which is a cycle formed by a new edge $e(v_i, v_j)$ and the subpath $P(v_i, v_j)$.

For any graph G used in our discussion (e.g., G is $G(i, j)$, $C(i, j)$, or P) and any two vertices v and v' of G, we use $d_G(v, v')$ to denote the length of any shortest path from v to v' in G and we also refer to $d_G(v, v')$ as the *distance* from v to v' in G. Following this definition, $d_P(v_i, v_j)$ is the length of the subpath $P(v_i, v_j)$ and $d_{C(i,j)}(v, v')$ is the distance between v to v' in the cycle $C(i, j)$. For any path π in G, we use $|\pi|$ to denote the length of π. We also use $|C(i, j)|$ to denote the total length of all edges of the cycle $C(i, j)$.

Our algorithm will frequently compute $d_P(v_i, v_j)$ for any index pair (i, j). This can be done in constant time after $O(n)$ time preprocessing, e.g., compute the prefix sum $d_P(v_1, v_k)$ for all $1 \leq k \leq n$.

For any vertex v of any graph G used in our discussion, a vertex v' of G is called a *farthest vertex* of v if it maximizes $d_G(v, v')$. A vertex v_c of G is called a *center* if its distance to its farthest vertex is minimized, and the distance from v_c to its farthest vertex is called the *radius* of G. Therefore, our problem is to find an index pair (i, j) such that the radius of $G(i, j)$ is minimized.

Let (i^*, j^*) denote an optimal solution (with $i^* < j^*$), i.e., $e(v_{i^*}, v_{j^*})$ is the new edge to be added. Let c^* denote the index of a center of $G(i^*, j^*)$, r^* the radius of $G(i^*, j^*)$, a^* the index of a farthest vertex of v_{c^*} in $G(i^*, j^*)$, and π^* a shortest path from v_{c^*} to v_{a^*} in $G(i^*, j^*)$. Note that the center of $G(i^*, j^*)$ may not be unique, in which case we use c^* to refer to an arbitrary one, but once c^* is fixed we will never change it throughout the paper. So as a^* and π^*. Note that $c^* \neq a^*$ since otherwise the graph would have only one vertex.

3 The Algorithm

As discussed in Section 1.2, we consider a constant number of configurations for the optimal solution $G(i^*, j^*)$. For each configuration, we compute in $O(n)$ time a candidate solution (consisting of an index pair (i', j'), a candidate center c' and a candidate radius r') such that if the optimal solution conforms to the configuration, then our candidate solution is an optimal one, i.e., $r^* = r'$. On the other hand, the candidate solution is a *feasible* one, i.e., the distances from c' to all vertices in $G(i', j')$ are at most r'.

In the following, we first give an overview of all configurations and then present algorithms to compute candidate solutions for them.

3.1 Configuration overview

The configurations are defined with respect to the locations of v_{a^*} and v_{c^*} as well as whether the path π^* contains the new edge $e(v_{i^*}, v_{j^*})$.

Depending on whether $c^* \in (i^*, j^*)$, there are two main cases.

Case 1: $c^* \notin (i^*, j^*)$**.** In this case, c^* is either in $[1, i^*]$ or in $[j^*, n]$. Hence, there are two subcases.

- **Case 1.1:** $c^* \in [1, i^*]$. See Fig. 3.
- **Case 1.2:** $c^* \in [j^*, n]$.
  This case is symmetric to Case 1.1.

Case 2: $c^* \in (i^*, j^*)$**.** Notice that a^* cannot be in $[2, i^*] \cup [j^*, n-1]$. Hence, there are three subcases $a^* = 1$, $a^* = n$, and $a^* \in (i^*, j^*)$.

- **Case 2.1:** $a^* = 1$.
  This case further has two subcases depending on whether the new edge $e(v_i^*, v_j^*)$ is contained in the path π^*.
  - **Case 2.1.1:** $e(v_i^*, v_j^*) \subseteq \pi^*$. See Fig. 5.
  - **Case 2.1.2:** $e(v_i^*, v_j^*) \not\subseteq \pi^*$. See Fig. 7.
- **Case 2.2:** $a^* = n$. This case is symmetric to Case 2.1.
- **Case 2.3:** $a^* \in (i^*, j^*)$.

In fact, we will only compute candidate solutions for Cases 1.1 and 1.2. We will show that other cases can be reduced to these two cases (i.e., if any case other than Case 1.1 and Case 1.2 has an optimal solution, then one of Case 1.1 and Case 1.2 must have an optimal solution).

3.2 Computing candidate solutions

We are now in a position to describe our algorithms for computing candidate solutions.

Case 1: $c^* \notin (i^*, j^*)$.

Depending on whether $c^* \in [1, i^*]$ or $c^* \in [j^*, n]$, there are two subcases.

Case 1.1: $c^* \in [1, i^*]$.

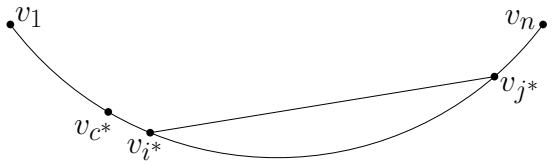


Figure 3: Illustrating Case 1.1: $c^* \in [1, i^*]$.

Refer to Fig. 3. In this case, either $a^* = 1$ or $a^* \in [i^*, n]$ and thus the radius r^* is equal to

$$\max\{d_P(v_1, v_{c^*}), d_P(v_{c^*}, v_{i^*}) + \max_{k\in[i^*,n]} d_{G(i^*,j^*)}(v_{i^*}, v_k)\}.$$

Definition 1 *For each $i \in [1, n-1]$, define*

$$\lambda_i = \min_{j\in[i,n]} \max_{k\in[i,n]} d_{G(i,j)}(v_i, v_k),$$

$$j_i = \arg\min_{j\in[i,n]} \max_{k\in[i,n]} d_{G(i,j)}(v_i, v_k),$$

$$r_i = \min_{k\in[1,i]} \max\{d_P(v_1, v_k), d_P(v_k, v_i) + \lambda_i\},$$

$$c_i = \arg\min_{k\in[1,i]} \max\{d_P(v_1, v_k), d_P(v_k, v_i) + \lambda_i\}.$$

The values λ_i and j_i were also used for solving the continuous problem in [12], where an algorithm was given that can compute λ_i and j_i for all $i = 1, 2, \ldots, n-1$ in $O(n)$ time. For our discrete problem, we also need to compute r_i and c_i for all $i = 1, 2, \ldots, n-1$. To this end, we propose an $O(n)$-time algorithm in Lemma 1.

Lemma 1 *The values r_i and c_i for all $i = 1, 2, \ldots, n-1$ can be computed in $O(n)$ time.*

Proof. We first compute λ_i for all $i = 1, 2, \ldots, n-1$ in $O(n)$ time [12]. Note that once c_i for all $i = 1, 2, \ldots, n-1$ are known, all r_i can be computed in additional $O(n)$ time because $r_i = \max\{d_P(v_1, v_{c_i}), d_P(v_{c_i}, v_i) + \lambda_i)\}$. Hence, we will focus on computing c_i below.

For each $i \in [1, n-1]$, define k_i as the largest index $k \in [1, i]$ such that $d_P(v_1, v_k) \le d_P(v_k, v_i) + \lambda_i$.

We claim that for each $i \in [1, n-1]$, c_i is either k_i or $k_i + 1$. Indeed, as k changes $[1, i]$, the value $d_P(v_1, v_k)$ is monotonically increasing while the value $d_P(v_k, v_i) + \lambda_i$ is monotonically decreasing. By the definition of c_i and k_i, the claim follows.

In light of the claim, once k_i is known, c_i can be determined in additional $O(1)$ time. In the following, we describe an algorithm to compute k_i for all $i = 1, 2, \ldots, n-1$ in $O(n)$ time.

We first prove a critical monotonicity property: $k_i \le k_{i+1}$ for all $i \in [1, n-2]$. To this end, it suffices to show that $d_P(v_1, v_{k_i}) \le d_P(v_{k_i}, v_{i+1}) + \lambda_{i+1}$. We claim that $\lambda_i \le d_P(v_i, v_{i+1}) + \lambda_{i+1}$. Before proving the claim, we use the claim to prove the monotonicity property:

$$\begin{aligned} d_P(v_1, v_{k_i}) &\le d_P(v_{k_i}, v_i) + \lambda_i \\ &= d_P(v_{k_i}, v_{i+1}) - d_P(v_i, v_{i+1}) + \lambda_i \\ &\le d_P(v_{k_i}, v_{i+1}) + \lambda_{i+1}. \end{aligned}$$

The first inequaltiy is due to the definition of k_i while the last inequality is due to the above claim. This proves the monotonicity property. In the following we prove the claim. The proof involves two graphs $G(i, j_{i+1})$ and $G(i+1, j_{i+1})$, e.g., see Fig. 4.

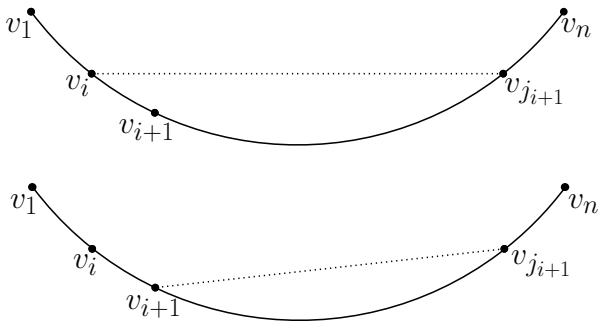


Figure 4: Illustrating the two graphs $G(i, j_{i+1})$ (top) and $G(i+1, j_{i+1})$ (bottom).

By definition, $\lambda_i \le \max_{k\in[i,n]} d_{G(i,j_{i+1})}(v_i, v_k)$. Define $k' = \arg\max_{k\in[i,n]} d_{G(i,j_{i+1})}(v_i, v_k)$. Hence, $\lambda_i \le d_{G(i,j_{i+1})}(v_i, v_{k'})$. It is not difficult to see that either $k' = n$ or $k' \in (i, j_{i+1})$. Below we prove $\lambda_i \le d_P(v_i, v_{i+1}) + \lambda_{i+1}$ for each case.

If $k' = n$, then due to the triangle inequality, $d_{G(i,j_{i+1})}(v_i, v_{k'}) = |v_i v_{j_{i+1}}| + d_P(v_{j_{i+1}}, v_n)$. Also due to the triangle inequality, $|v_i v_{j_{i+1}}| \le d_P(v_i, v_{i+1}) + |v_{i+1} v_{j_{i+1}}|$ and $d_{G(i+1,j_{i+1})}(v_{i+1}, v_n) = |v_{i+1} v_{j_{i+1}}| + d_P(v_{j_{i+1}}, v_n)$. In addition, according to the definition of λ_{i+1}, we have $\lambda_{i+1} = \max_{k\in[i+1,n]} d_{G(i+1,j_{i+1})}(v_{i+1}, v_k) \ge d_{G(i+1,j_{i+1})}(v_{i+1}, v_n)$. Combining all above, we can derive

$$\begin{aligned} \lambda_i &\le d_{G(i,j_{i+1})}(v_i, v_{k'}) = d_{G(i,j_{i+1})}(v_i, v_n) \\ &= |v_i v_{j_{i+1}}| + d_P(v_{j_{i+1}}, v_n) \\ &\le d_P(v_i, v_{i+1}) + |v_{i+1} v_{j_{i+1}}| + d_P(v_{j_{i+1}}, v_n) \\ &= d_P(v_i, v_{i+1}) + d_{G(i+1,j_{i+1})}(v_{i+1}, v_n) \\ &\le d_P(v_i, v_{i+1}) + \max_{k\in[i+1,n]} d_{G(i+1,j_{i+1})}(v_{i+1}, v_k) \\ &= d_P(v_i, v_{i+1}) + \lambda_{i+1}. \end{aligned}$$

We proceed to the case $k' \in (i, j_{i+1})$. Consider the graph $G(i+1, j_{i+1})$. $d_{G(i+1,j_{i+1})}(v_{i+1}, v_{k'})$ is equal to either $d_P(v_{i+1}, v_{k'})$ or $|v_{i+1}v_{j_{i+1}}| + d_P(v_{k'}, v_{j_{i+1}})$.

In the former case, we have

$$\begin{aligned}\lambda_i &\le d_{G(i,j_{i+1})}(v_i, v_{k'}) \le d_P(v_i, v_{k'}) \\ &= d_P(v_i, v_{i+1}) + d_P(v_{i+1}, v_{k'}) \\ &= d_P(v_i, v_{i+1}) + d_{G(i+1,j_{i+1})}(v_{i+1}, v_{k'}) \\ &\le d_P(v_i, v_{i+1}) + \max_{k\in[i+1,n]} d_{G(i+1,j_{i+1})}(v_{i+1}, v_k) \\ &= d_P(v_i, v_{i+1}) + \lambda_{i+1}.\end{aligned}$$

In the latter case, similarly we can derive

$$\begin{aligned}\lambda_i &\le d_{G(i,j_{i+1})}(v_i, v_{k'}) \le |v_i v_{j_{i+1}}| + d_P(v_{k'}, v_{j_{i+1}}) \\ &\le d_P(v_i, v_{i+1}) + |v_{i+1}v_{j_{i+1}}| + d_P(v_{k'}, v_{j_{i+1}}) \\ &= d_P(v_i, v_{i+1}) + d_{G(i+1,j_{i+1})}(v_{i+1}, v_{k'}) \\ &\le d_P(v_i, v_{i+1}) + \max_{k\in[i+1,n]} d_{G(i+1,j_{i+1})}(v_{i+1}, v_k) \\ &= d_P(v_i, v_{i+1}) + \lambda_{i+1}.\end{aligned}$$

This proves the claim and thus the monotonicity property of k_i's.

Using the monotonicity property of k_i's, we can easily compute all k_i's in $O(n)$ time as follows. Starting from $i = 1$, the algorithm incrementally computes k_i for all $i = 1, 2, \ldots, n-1$. The algorithm maintains an index k. Initially, $k = i = 1$ and $k_i = k$. Consider a general step where k_i has just been computed and $k = k_i$. Next we compute k_{i+1} as follows. As long as $d_P(v_1, v_{k+1}) \le d_P(v_{k+1}, v_{i+1}) + \lambda_{i+1}$, we increment k by one. After that, we set $k_{i+1} = k$. The monotonicity property of k_i's guarantees the correctness of the algorithm. The running time is $O(n)$.

The lemma is thus proved. □

We obtain a candidate solution for this configuration as follows. We first compute λ_i and j_i for all $i = 1, 2, \ldots, n-1$ in $O(n)$ time [12]. We then use Lemma 1 to compute r_i and c_i for all $i = 1, 2, \ldots, n-1$. Let $i' = \arg\min_{i\in[1,n-1]} r_i$. Let $r' = r_{i'}$ and $j' = j_{i'}$. We return (i', j'), c', and r' as a candidate solution for this configuration. Notice that the candidate solution is a feasible solution, i.e., the distances from $v_{c'}$ to all vertices in $G(v_{i'}, v_{j'})$ are at most r'. The following lemma establishes the correctness of our candidate solution.

Lemma 2 $r' = r^*$.

Proof. First of all, as the candidate solution is a feasible one, by the definition of r^*, $r^* \le r'$ holds. It remains to prove $r' \le r^*$.

Recall that $r^* = \max\{d_P(v_{c^*}, v_1), d_P(v_{c^*}, v_{i^*}) + \max_{k\in[i^*,n]} d_{G(i^*,j^*)}(v_{i^*}, v_k)\}$. By the definition of λ_i, it holds that $\lambda_{i^*} \le \max_{k\in[i^*,n]} d_{G(i^*,j^*)}(v_{i^*}, v_k)$. Thus, $r^* \ge \max\{d_P(v_{c^*}, v_1), d_P(v_{c^*}, v_{i^*}) + \lambda_{i^*}\}$. We claim that $r^* = \max\{d_P(v_{c^*}, v_1), d_P(v_{c^*}, v_{i^*}) + \lambda_{i^*}\}$. Indeed, the value $\max\{d_P(v_{c^*}, v_1), d_P(v_{c^*}, v_{i^*}) + \lambda_{i^*}\}$ is equal to the distance from vertex c^* to its farthest vertex in the graph $G(i^*, j_{i^*})$. By the definition of r^*, $r^* \le \max\{d_P(v_{c^*}, v_1), d_P(v_{c^*}, v_{i^*}) + \lambda_{i^*}\}$. The claim thus follows.

The claim and the definition of r_{i^*} together lead to $r_{i^*} \le r^*$. Further, by the definition of the index i', we have $r' = r_{i'} \le r_{i^*} \le r^*$. The lemma thus follows. □

Case 1.2: $c^* \in [j^*, n]$.

This case is symmetric to Case 1.1 and we use a similar algorithm to compute a candidate solution. The details are omitted but can be found in the full paper [17].

Case 2: $c^* \in (i^*, j^*)$.

We now consider the case $c^* \in (i^*, j^*)$. In this case, it is easy to see that $d_{G(i^*,j^*)}(v_{c^*}, v_k) < d_{G(i^*,j^*)}(v_{c^*}, v_1)$ for any $k \in (1, i^*]$ and similarly $d_{G(i^*,j^*)}(v_{c^*}, v_k) < d_{G(i^*,j^*)}(v_{c^*}, v_n)$ for any $k \in [j^*, n)$. Hence, a^* cannot be in $(1, i^*] \cup [j^*, n)$. Thus, $a^* = 1$, $a^* = n$, or $a^* \in (i^*, j^*)$.

Case 2.1: $a^* = 1$.

Depending on whether the new added edge $e(v_{i^*}, v_{j^*})$ is contained in the path π^*, there are two cases.

Case 2.1.1: $e(v_{i^*}, v_{j^*}) \subseteq \pi^*$.

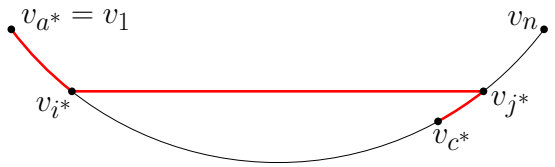


Figure 5: Illustrating the configuration for Case 2.1.1, where $c^* \in (i^*, j^*)$, $a^* = 1$, and $e(v_{i^*}, v_{j^*}) \subseteq \pi^*$. The thick (red) path is π^*.

In this case, $c^* \in (i^*, j^*)$, $a^* = 1$, and $e(v_{i^*}, v_{j^*}) \subseteq \pi^*$. This implies that $\pi^* = P(v_{c^*}, v_{j^*}) \cup e(v_{i^*}, v_{j^*}) \cup P(v_1, v_{i^*})$; e.g., see Fig. 5.

Lemma 3 *The index pair (i^*, c^*) is an optimal solution and c^* is a center of the graph $G(i^*, c^*)$.*

Proof. We show that the distances from c^* to all vertices in the graph $G(i^*, c^*)$ are at most r^* (e.g., see Fig. 6). This implies that the radius of $G(i^*, c^*)$ is at most r^* and thus proves the lemma.

Let k be any index in $[1, n]$. Our goal is to prove $d_{G(i^*,c^*)}(v_{c^*}, v_k) \le r^*$.

If $k \in [1, i^*]$, then $d_{G(i^*,c^*)}(v_{c^*}, v_k) \le |e(v_{i^*}, v_{c^*})| + d_P(v_k, v_{i^*})$. By the triangle inequality, $|e(v_{i^*}, v_{c^*})| \le$

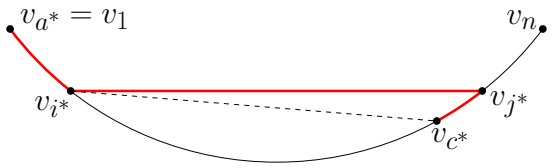


Figure 6: Illustrating Lemma 3: The distances from c^* to all vertices in $G(i^*, c^*)$ are at most r^*.

$|e(v_{i^*}, v_{j^*})| + d_P(v_{c^*}, v_{j^*})$ holds. Hence, $r^* = |\pi^*| = |e(v_{i^*}, v_{j^*})| + d_P(v_{c^*}, v_{j^*}) + d_P(v_1, v_{i^*}) \geq |e(v_{i^*}, v_{c^*})| + d_P(v_1, v_{i^*}) \geq |e(v_{i^*}, v_{c^*})| + d_P(v_k, v_{i^*}) \geq d_{G(i^*,c^*)}(v_{c^*}, v_k)$.

If $k \in [c^*, n]$, then $d_{G(i^*,c^*)}(v_{c^*}, v_k) \leq d_P(v_{c^*}, v_k) \leq d_P(v_{c^*}, v_n)$. As π^* is a shortest path in $G(i^*, j^*)$ and π^* contains $P(v_{c^*}, v_{j^*})$, $P(v_{c^*}, v_n)$ must be a shortest path from v_{c^*} to v_n in $G(i^*, j^*)$, implying that $d_P(v_{c^*}, v_n) \leq r^*$. Therefore, $d_{G(i^*,c^*)}(v_{c^*}, v_k) \leq r^*$ holds.

If $k \in (i^*, c^*)$, then both v_{c^*} and v_k are in the cycle $C(i^*, j^*)$ of the graph $G(i^*, j^*)$ and are also in the cycle $C(i^*, c^*)$ of the graph $G(i^*, c^*)$. Hence, $d_{G(i^*,j^*)}(v_{c^*}, v_k) = d_{C(i^*,j^*)}(v_{c^*}, v_k)$ and $d_{G(i^*,c^*)}(v_{c^*}, v_k) = d_{C(i^*,c^*)}(v_{c^*}, v_k)$. Due to the triangle inequality, $|C(i^*, c^*)| \leq |C(i^*, j^*)|$. Hence, $d_{C(i^*,c^*)}(v_{c^*}, v_k) \leq d_{C(i^*,j^*)}(v_{c^*}, v_k)$. As $d_{G(i^*,j^*)}(v_{c^*}, v_k) \leq r^*$, we can now obtain $d_{G(i^*,c^*)}(v_{c^*}, v_k) = d_{C(i^*,c^*)}(v_{c^*}, v_k) \leq d_{C(i^*,j^*)}(v_{c^*}, v_k) = d_{G(i^*,j^*)}(v_{c^*}, v_k) \leq r^*$. □

Because (i^*, c^*) is an optimal solution with c^* as a center in the graph $G(i^*, c^*)$, it is a configuration of Case 1.2. Hence, the candidate solution found by our algorithm for Case 1.2 is also an optimal solution. Thus, it is not necessary to compute a candidate solution for this case any more, i.e., this case is reduced to Case 1.2.

Case 2.1.2: $e(v_{i^*}, v_{j^*}) \not\subseteq \pi^*$.

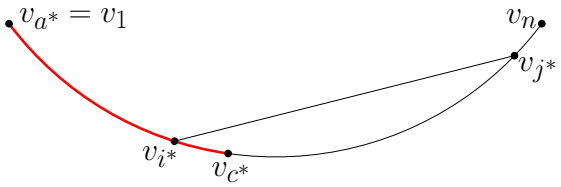


Figure 7: Illustrating the configuration for Case 2.1.2, where $c^* \in (i^*, j^*)$, $a^* = 1$, and $e(v_{i^*}, v_{j^*}) \not\subseteq \pi^*$. The thick (red) path is π^*.

Refer to Fig. 7. In this case, $c^* \in (i^*, j^*)$, $a^* = 1$, and $e(v_{i^*}, v_{j^*}) \not\subseteq \pi^*$. This implies that $\pi^* = P(v_1, v_{c^*})$. The following lemma reduces this case to Case 1.1.

Lemma 4 *The index pair (c^*, j^*) is an optimal solution and c^* is a center of the graph $G(c^*, j^*)$.*

Proof. Some proof techniques are similar to Lemma 3. It suffices to show that the distances from c^* to all vertices in $G(c^*, j^*)$ are at most r^*. Let k be any index in $[1, n]$. Our goal is to prove $d_{G(c^*,j^*)}(v_{c^*}, v_k) \leq r^*$.

Note that $d_P(v_{c^*}, v_1) = r^*$, for $\pi^* = P(v_1, v_{c^*})$.

If $k \in [1, c^*]$, then $d_{G(c^*,j^*)}(v_{c^*}, v_k) \leq d_P(v_{c^*}, v_k) \leq d_P(v_{c^*}, v_1) = r^*$.

If $k \in [j^*, n]$, then $d_{G(c^*,j^*)}(v_{c^*}, v_k) \leq d_{G(c^*,j^*)}(v_{c^*}, v_n)$. Below we prove $d_{G(c^*,j^*)}(v_{c^*}, v_n) \leq d_{G(i^*,j^*)}(v_{c^*}, v_n)$, which is at most r^*. Note that $d_{G(i^*,j^*)}(v_{c^*}, v_n) = \min\{d_P(v_{c^*}, v_n), d_P(v_{c^*}, v_{i^*}) + |e(v_{i^*}, v_{j^*})| + d_P(v_{j^*}, v_n)\}$. If $d_{G(i^*,j^*)}(v_{c^*}, v_n) = d_P(v_{c^*}, v_n)$, then we have $d_{G(c^*,j^*)}(v_{c^*}, v_n) \leq d_P(v_{c^*}, v_n) = d_{G(i^*,j^*)}(v_{c^*}, v_n)$. If $d_{G(i^*,j^*)}(v_{c^*}, v_n) = d_P(v_{c^*}, v_{i^*}) + |e(v_{i^*}, v_{j^*})| + d_P(v_{j^*}, v_n)$, then by the triangle inequality, $d_{G(c^*,j^*)}(v_{c^*}, v_n) \leq |e(v_{c^*}, v_{j^*})| + d_P(v_{j^*}, v_n) \leq d_P(v_{c^*}, v_{i^*}) + |e(v_{i^*}, v_{j^*})| + d_P(v_{j^*}, v_n) = d_{G(i^*,j^*)}(v_{c^*}, v_n)$.

If $k \in (c^*, j^*)$, then both v_{c^*} and v_k are in the cycle $C(i^*, j^*)$ of the graph $G(i^*, j^*)$ and are also in the cycle $C(c^*, j^*)$ of the graph $G(c^*, j^*)$. By a similar analysis as that for Lemma 3, we can obtain $d_{G(c^*,j^*)}(v_{c^*}, v_k) \leq d_{G(i^*,j^*)}(v_{c^*}, v_k) \leq r^*$. □

Case 2.2: $a^* = n$.

This case is symmetric to Case 2.1 and we omit the details.

Case 2.3: $a^* \in (i^*, j^*)$.

In this case, both a^* and c^* are in (i^*, j^*). Without loss of generality, we assume that $c^* < a^*$. The following lemma reduces this case to Case 1.1. Due to the space limit, the proof is omitted but can be found in the full paper [17].

Lemma 5 *The index pair (c^*, j^*) is an optimal solution and c^* is a center of the graph $G(c^*, j^*)$.*

Summary. We have computed a candidate solution for each of Case 1.1 and Case 1.2. Each candidate solution is also a feasible one. We have proved that if an optimal solution belongs to one of the two cases, then the corresponding candidate solution must also be an optimal solution. On the other hand, we have shown that other cases can be reduced to the two cases. Therefore, one of the two candidate solutions must be an optimal one. As a final step of our algorithm, among the two candidate solutions, we return the one with smaller candidate radius as our optimal solution. The running time of the entire algorithm is $O(n)$.

Theorem 6 *The discrete-ROAP problem can be solved in linear time.*

References

[1] N. Alon, A. Gyárfás, and M. Ruszinkó. Decreasing the diameter of bounded degree graphs. *Journal of Graph Theory*, 35:161–172, 2000.

[2] S. Bae, M. de Berg, O. Cheong, J. Gudmundsson, and C. Levcopoulos. Shortcuts for the circle. In *Proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC)*, pages 9:1–9:13, 2017.

[3] D. Bilò. Almost optimal algorithms for diameter-optimally augmenting trees. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC)*, pages 40:1–40:13, 2018.

[4] D. Bilò, L. Gualà, and G. Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theoretical Computer Science*, 417:12–22, 2012.

[5] J.-L. De Carufel, C. Grimm, A. Maheshwari, and M. Smid. Minimizing the continuous diameter when augmenting paths and cycles with shortcuts. In *Proceedings of the 15th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 27:1–27:14, 2016.

[6] J.-L. De Carufel, C. Grimm, S. Schirra, and M. Smid. Minimizing the continuous diameter when augmenting a tree with a shortcut. In *Proceedings of the 15th Algorithms and Data Structures Symposium (WADS)*, pages 301–312, 2017.

[7] F. Frati, S. Gaspers, J. Gudmundsson, and L. Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, 72:995–1010, 2015.

[8] Y. Gao, D. Hare, and J. Nastos. The parametric complexity of graph diameter augmentation. *Discrete Applied Mathematics*, 161:1626–1631, 2013.

[9] U. Große, J. Gudmundsson, C. Knauer, M. Smid, and F. Stehn. Fast algorithms for diameter-optimally augmenting paths. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 678–688, 2015.

[10] U. Große, J. Gudmundsson, C. Knauer, M. Smid, and F. Stehn. Fast algorithms for diameter-optimally augmenting paths and trees. *arXiv:1607.05547*, 2016.

[11] T. Ishii. Augmenting outerplanar graphs to meet diameter requirements. *Journal of Graph Theory*, 74:392–416, 2013.

[12] C. Johnson and H. Wang. A linear-time algorithm for radius-optimally augmenting paths in a metric space. In *Proceedings of the 16th Algorithms and Data Structures Symposium (WADS)*, pages 466–480, 2019.

[13] C.-L. Li, S. McCormick, and D. Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Operations Research Letters*, 11:303–308, 1992.

[14] E. Oh and H.-K. Ahn. A near-optimal algorithm for finding an optimal shortcut of a tree. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 59:1–59:12, 2016.

[15] A. Schoone, H. Bodlaender, and J. V. Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11:409–427, 1997.

[16] H. Wang. An improved algorithm for diameter-optimally augmenting paths in a metric space. *Computational Geometry: Theory and Applications*, 75:11–21, 2018.

[17] H. Wang and Y. Zhao. A linear-time algorithm for discrete radius optimally augmenting paths in a metric space. *arXiv:2006.14093*, 2020.

Topological and geometric methods for graph analysis

Yusu Wang*

In recent years, topological and geometric data analysis (TGDA) has emerged as a new and promising field for processing, analyzing and understanding complex data. Indeed, geometry and topology form natural platforms for data analysis, with geometry describing the "shape" and "structure" behind data; and topology characterizing / summarizing both the domain where data are sampled from, as well as functions and maps associated to them.

In this talk, I will show how topological and geometric ideas can be used to analyze graph data, which occurs ubiquitously across science and engineering. Graphs could be geometric in nature, such as road networks in GIS, or relational and abstract. I will particularly focus on the reconstruction of hidden geometric graphs from noisy data, as well as graph matching and classification. I will discuss the motivating applications, algorithm development, and theoretical guarantees for these methods. Through these topics, I aim to illustrate the important role that geometric and topological ideas can play in data analysis.

*University of California, San Diego, yusuwang@ucsd.edu

Computing the Carathéodory Number of a Point

Sergey Bereg* Mohammadreza Haghpanah*

Abstract

Carathéodory's theorem says that any point in the convex hull of a set P in $\mathbb{R}^d$ is in the convex hull of a subset P' of P such that $|P'| \leq d+1$. For some sets P, the upper bound $d+1$ can be improved. The best upper bound for P is known as the *Carathéodory number* [2, 15, 17]. In this paper, we study a computational problem of finding the smallest set P' for a given set P and a point p. We call the size of this set P', the *Carathéodory number of a point p* or CNP. We show that the problem of deciding the Carathéodory number of a point is NP-hard. Furthermore, we show that the problem is k-LDT-hard. We present two algorithms for computing a smallest set P', if CNP$= 2, 3$.

Bárány [1] generalized Carathéodory's theorem by using $d+1$ sets (colored sets) such that their convex hulls intersect. We introduce a Colorful Carathéodory number of a point or CCNP which can be smaller than $d+1$. Then we extend our results for CNP to CCNP.

1 Introduction

The well-known Carathéodory's theorem deals with the convex hull of a set P, denoted by $\mathsf{conv}(P)$.

Theorem 1 (Carathéodory's theorem [8, 13]) *Let P be a set of points in $\mathbb{R}^d$ and p be a point in $\mathsf{conv}(P)$. Then there is a subset P' of P consisting of at most $d+1$ points such that $p \in \mathsf{conv}(P')$.*

Sometimes there is a set P' of smaller size such that $p \in \mathsf{conv}(P')$, see Figure 1 for an example. We define a *Carathéodory number of a point.*

Definition 2 *For a set of points $P \subset \mathbb{R}^d$ and a point $p \in \mathsf{conv}(P)$,* Carathéodory number of p with respect to P, *denoted by $C(P,p)$, is the smallest integer k such that $p \in \mathsf{conv}(P')$ for a subset $P' \subseteq P$ of size k.*

Carathéodory's theorem guarantees that for every set of points $P \subset \mathbb{R}^d$ and $p \in \mathsf{conv}(P)$, $C(P,p)$ is well-defined and $C(P,p) \leq d+1$. This is related to the well-known concept of the *Carathéodory number of a set* that is the smallest integer k such that, for any point $p \in \mathsf{conv}(P)$, there is a subset P' of P consisting of at most k points such that $p \in \mathsf{conv}(P')$. Equivalently, it can be defined using $C(P,p)$ as follows.

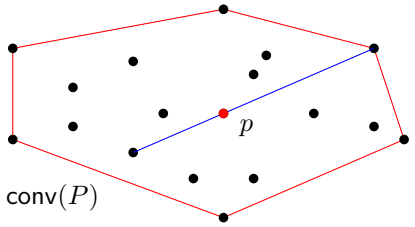


Figure 1: Point $p \in \mathsf{conv}(P)$ with $C(P,p) = 2$.

Definition 3 *For a set of points $P \subset \mathbb{R}^d$,* Carathéodory number of P, *denoted by $C(P)$, is the largest integer k where there exists a point $p \in \mathsf{conv}(P)$ such that $C(P,p) = k$.*

The Carathéodory number of a set is being studied for more than 90 years [15, 17], in a more general setting. The Carathéodory number of any set $P \subset \mathbb{R}^d$ is at most $d+1$ by Carathéodory's theorem. For a compactum $P \subset \mathbb{R}^d$, Bárány and Karasev [2] found sufficient conditions to have Carathéodory number less than $d+1$. Kay and Womble [22] showed a relation between the Carathéodory, Helly, and Radon numbers. Recently, Ito and Lourenço [19] showed an upper bound for the Carathéodory number of a set. Much research has been done on the Carathéodory number for some specific sets. Sierksma [27] studied the Carathéodory number for convex-product-structures, Naldi [25] for Hilbert cones of quadratic forms and binary forms. Bui and Karasev [7] showed the Carathéodory number for arbitrary gauge set K in $\mathbb{R}^d$ is greater than $d-1$. Also, the Carathéodory number for several graph convexities is studied in graph theory [4, 11, 12].

In this paper, we are interested in computing the Carathéodory number of a point. We found the following characterization of the Carathéodory number of a set in $\mathbb{R}^d$. This characterization of the Carathéodory number could be known but we were not able to find it[1]. Recall that the *affine hull* of a set S is the smallest affine

*University of Texas at Dallas, Richardson, TX 75080, USA. {besp,Mohammadreza.Haghpanah}@utdallas.edu

[1] We found that the upper bound of the Carathéodory number of a set follows from Proposition 1.15(ii) [28], see the proof in Section 2.

set containing S (a set A is *affine* if, for any $a, b \in A$, the line passing through a and b is also contained in A). We denote it by $\mathsf{aff}(S)$. The dimension of an affine set S, denoted by $\dim(S)$ is the dimension of its underlying linear subspace.

Proposition 4 *The Carathéodory number of any non-empty set $P \subseteq \mathbb{R}^d$ is equal to* $\dim(\mathsf{aff}(P)) + 1$.

The Carathéodory number of a finite set in $\mathbb{R}^d$ can be computed using Proposition 4. In this paper, we study the computational problem of finding the Carathéodory number of a point with respect to a finite set.

Problem 5 (ComputingCNP)

Given *a set of points P in $\mathbb{R}^d$ and a point $p \in \mathsf{conv}(P)$.*

Compute *a subset P' of P such that (i) $p \in \mathsf{conv}(P')$ and (ii) the size of P' is minimized.*

We show that the decision version of ComputingCNP is NP-hard if the dimension d is part of the input. Furthermore, we show that the problem is k-LDT-hard if dimension d is fixed. We present two algorithms for ComputingCNP when $C(P, p) = 2, 3$.

Bárány [1] generalized Carathéodory's theorem by using $d+1$ sets (colored sets) such that their convex hulls intersect. As in [24], we call these sets *color classes* and we call a set of $d+1$ elements, one from each color class, a *colorful choice*.

Theorem 6 (Colorful Carathéodory theorem [1]) *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_{d+1}\}$ be a collection of sets of points in $\mathbb{R}^d$ and p be a point such that $p \in \cap_{i=1}^{d+1} \mathsf{conv}(P_i)$. Then there is a colorful choice P' such that $p \in \mathsf{conv}(P')$.*

It is known that the number of color classes in Theorem 6 cannot be reduced. One may ask whether the number of colors in set P' can be reduced. Sometimes there is a set P' of size smaller than $d+1$ such that $p \in \mathsf{conv}(P')$, see Figure 2 for an example. In this paper, we define a *Colorful Carathéodory number of a point*. We call a set of at most $d+1$ elements, one from color class, a *rainbow*, i.e. a rainbow is a subset of a colorful choice for $\mathcal{P}$. We use notation $[k] = \{1, 2, \ldots, k\}$.

Definition 7 *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_{d+1}\}$ be a collection of sets of points in $\mathbb{R}^d$ and p be a point such that $p \in \cap_{i=1}^{d+1} \mathsf{conv}(P_i)$. The* Colorful Carathéodory number *of p with respect to $\mathcal{P}$, denoted by $CC(\mathcal{P}, p)$, is the smallest size of a rainbow P' for $\mathcal{P}$ such that $p \in \mathsf{conv}(P')$.*

The colorful Carathéodory theorem guarantees that $CC(\mathcal{P}, p) \leq d + 1$. In this paper, we also propose to study a new problem of computing $CC(\mathcal{P}, p)$.

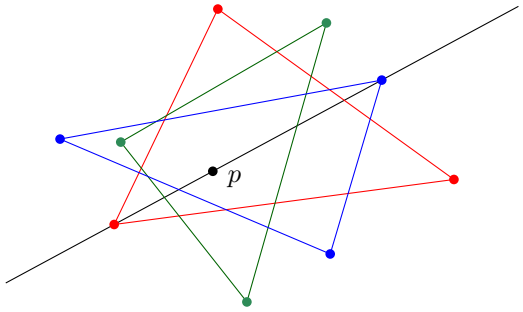


Figure 2: Three sets capturing p in the plane. There is a 2-colorful choice using one red point and one blue point.

Problem 8 (ComputingCCNP)

Given *a collection $\mathcal{P} = \{P_1, P_2, \cdots, P_{d+1}\}$ of sets of points in $\mathbb{R}^d$ and a point $p \in \cap_{i=1}^{d+1} \mathsf{conv}(P_i)$.*

Compute *a rainbow P' of the smallest size such that $p \in \mathsf{conv}(P')$.*

Related work. Bárány and Onn [3] describe an approximation algorithm to find a colorful set P' such that point p has distance at most ϵ from $\mathsf{conv}(P')$. Barman [6] showed that a rainbow P' of size $O(\gamma^2/\epsilon^2)$ for $\gamma = \max_{x \in P} \|x\|$ such that the distance between p and $\mathsf{conv}(P')$ is at most ϵ. Mulzer and Stein [24] studied a different approximation using m-colorful sets. A set P' is *m-colorful* if $P_i \cap P' \leq m$ for each color set P_i. Mulzer and Stein [24] give a polynomial algorithm to find a $\lceil \epsilon d \rceil$-colorful choice P' such that $p \in \mathsf{conv}(P')$ for some fixed $\epsilon > 0$. Meunier *et al.* [23] show that the problem of finding a colorful choice is $PPAD$ and PLS.

ComputingCNP is related to the sparse linear regression problem [18, 26] where a $d \times n$ matrix M and a vector $q \in \mathbb{R}^d$ are given and the task is to find a k-sparse vector τ minimizing $\|q - M\tau\|_2$. Natarajan [26] proved NP-hardness of this problem. Har-Peled, Indyk and Mahabadi [18] presented an algorithm with $n^{k-1}S(n, d, \varepsilon)$ space and $n^{k-1}T_Q(n, d, \varepsilon)$ query time where $S(n, d, \varepsilon)$ denotes the preprocessing time and space used by a $(1+\varepsilon)$-ANN (approximate nearest-neighbor) data-structure, and $T_Q(n, d, \varepsilon)$ denotes the query time. Recently, Cardinal and Ooms [9] studied the sparse regression problem and found a $O(n^{k-1} \log^{d-k+2} n)$-time randomized $(1+\varepsilon)$-approximation algorithm for this problem with d and ε constant.

Our results can be summarized as follows.

- We characterize the Carathéodory number of a finite set of distinct points in $\mathbb{R}^d$ (Proposition 4).
- We introduce new problems ComputingCNP and ComputingCCNP for computing $C(P, p)$ and $CC(\mathcal{P}, p)$. We show that DecidingCNP, the decision version of ComputingCNP, is

  - NP-hard (Theorem 10) if the dimension d is part of the input,
  - is k-LDT-hard if dimension d is fixed (Theorem 13).

- We present two algorithms in Section 4 for ComputingCNP when $C(P,p) = 2, 3$.
- Then we extend our results for ComputingCNP to ComputingCCNP in Section 5.

2 Proof of Proposition 4

Let P be a finite set of distinct points in $\mathbb{R}^d$. Theorem 4 states that

$$C(P) = \dim(\mathsf{aff}(P)) + 1.$$

Let $d' = \dim(\mathsf{aff}(P))$.

First, we will prove that $C(P) \geq d'+1$. There exists a set Q of $d'+1$ points of P which are affinely independent. Then $S = \mathsf{conv}(Q)$ is the $(d'+1)$-dimensional simplex. Consider the set A defined as

$$A = \bigcup_{P' \subset P, |P'| = d'} \mathsf{aff}(P').$$

Set A is the union of $\binom{n}{d'}$ sets each of dimension smaller than d'. Therefore $A \cap S \neq S$. For any point $p \in S \setminus A$, we have $C(P,p) \geq d'+1$. Therefore $C(P) \geq d'+1$.

Second, we show that $C(P) \leq d'+1$. This follows from Proposition 1.15(ii) [28] if we write n points of P as a $d \times n$ matrix X.

Proposition. Let $X \in \mathbb{R}^{d \times n}$ and $x \in \mathbb{R}^d$. If $x \in \mathsf{conv}(X)$, then $x \in \mathsf{conv}(X')$ holds for a subset $X' \subseteq X$ of at most $\mathsf{rank}\binom{\mathbb{1}}{X} = \dim(\mathsf{conv}(X)) + 1$ vectors in X.

3 Hardness of ComputingCNP

First, we state the decision problem corresponding to ComputingCNP.

Problem 9 (DecidingCNP)

Given *a set of points $P \subset \mathbb{R}^d$, a point $p \in \mathsf{conv}(P)$ and an integer $k \leq |P|$.*

Decide *whether $C(P,p) \leq k$.*

Observe that DecidingCNP can be solved in polynomial time if dimension d is a constant. We show that it is NP-hard if d is part of the input.

Theorem 10 DecidingCNP *is NP-hard.*

Proof. We reduce the following problem to DecidingCNP.

Problem 11 (ExactCoverBy3-Sets)

Given *a set $X = \{1, 2, 3, \ldots, m\}$ such that 3 is a divisor of m and a collection $\mathcal{S} = \{T_1, T_2, \ldots, T_n\}$ where $T_i \subset X$ and $|T_i| = 3$, for $1 \leq i \leq n$.*

Decide *whether there exists a subset $\mathcal{S}'$ of $\mathcal{S}$ such that $\mathcal{S}'$ is a partition of X, i.e. sets in $\mathcal{S}'$ are disjoint and their union is X.*

Problem ExactCoverBy3-Sets is a variant of ExactCover [21]. This problem is also known to be NP-complete [20].

For an instance of ExactCoverBy3-Sets, a set $X = \{1, 2, 3, \ldots, m\}$ such that 3 is a divisor of m and a collection $\mathcal{S} = \{T_1, T_2, \ldots, T_n\}$ where $T_i \subset X$ and $|T_i| = 3$ for $1 \leq i \leq n$, we construct an instance of DecidingCNP as follows. Set $k = m/3$, $p = (1, 1, \ldots, 1) \in \mathbb{R}^m$ and $P = \{p_1, p_2, \ldots, p_n\}$ where $p_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,m}) \in \mathbb{R}^m$ and

$$p_{i,j} = \begin{cases} k & \text{if } j \in T_i, \\ 0 & \text{otherwise.} \end{cases}$$

We show that there exists an exact cover for set X if and only if there exists a subset $P' \subset P$ of size k where $p \in \mathsf{conv}(P')$.

$\Longrightarrow$) Suppose that $\mathcal{S}'$ is a partition for X. Then for every $j \in X$ there exists unique $T_i \in \mathcal{S}'$ with $j \in T_i$. Set P' as the set of all points p_i such that $T_i \in \mathcal{S}'$. For any $j \in [m]$, there is exactly one point $p_i \in P'$ with $p_{i,j} = k$ and $p_{i',j} = 0$ for all other points $p_{i'} \in P'$. Therefore the j-th coordinate of $\sum_{p_i \in P'} p_i$ is equal to k and $\sum_{p_i \in P'} p_i = kp$. Hence, $p \in \mathsf{conv}(P')$.

$\Longleftarrow$) Suppose that $p \in \mathsf{conv}(P')$, i.e. $\sum_{p_i \in P'} \lambda_i p_i = p$. Then each $\lambda_i \leq \frac{1}{k}$, otherwise some coordinate of $\sum_{p_i \in P'} \lambda_i p_i$ is greater than 1. We have $\sum_{p_i \in P'} \lambda_i = 1$ and each $\lambda_i \geq 0$. Since $|P'| = k$, each λ_i must be equal to $1/k$. Let $\mathcal{S}'$ be the set of all T_i such that $p_i \in P'$. Then, $\mathcal{S}'$ is a partition of X. □

Now, suppose that the dimension d is fixed. We show that DecidingCNP is k-LDT-hard.

Problem 12 (k-LDT)

Given *a set of $A \subset \mathbb{R}$ and a k-variate linear function $\phi(x_1, x_2, \ldots, x_k) = \alpha_0 + \sum_{i=1}^{k} \alpha_i x_i$ where $\alpha_0, \alpha_1, \ldots, \alpha_k \in \mathbb{R}$.*

Decide *whether there exists $x = (x_1, x_2, \ldots, x_k) \in A^k$ where x is a root of ϕ.*

k-LDT-hardness implies k-SUM-hardness and many problems are known to be k-SUM-hard, see for example [5, 16]. Erickson [14] proved any algorithm in r-linear decision tree model for k-LDT problem has $\Omega(n^{\lceil \frac{k}{2} \rceil})$ time complexity.

Theorem 13 DecidingCNP *for a fixed dimension d is k-LDT-hard.*

Proof. We show a linear-time reduction of k-LDT to DecidingCNP. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of real numbers and linear function $\phi(x_1, x_2, \ldots, x_k) = \alpha_0 + \sum_{i=0}^{t} \alpha_i x_i$ be an instance of k-LDT problem. An instance of DecidingCNP must contain a set P, a point p, and an integer k' (it could be different from k in k-LDT). We construct an instance of DecidingCNP. We choose $k' = k + 1$. We construct set P as follows.

Let $\{e_1, e_2, \ldots, e_{k+1}\}$ be the standard basis of $\mathbb{R}^{k+1}$, i.e. $e_i = (e_{i,1}, e_{i,2}, \ldots, e_{i,k+1})$ where $e_{i,j} = 1$ if $j = i$ and $e_{i,j} = 0$ otherwise. For every $x_i \in A$, $1 \le i \le n$, we construct k points in $\mathbb{R}^{k+1}$, $y_{i,j}$, for $1 \le j \le k$, as follows

$$y_{i,j} = e_j + \alpha_j x_i e_{k+1}.$$

We also define $p = (-\alpha_0 e_{k+1} + \sum_{i=1}^{k} e_i)/k$.

$\Longrightarrow$) Suppose there is k integer $i_1, i_2, \ldots, i_k$ where $1 \le i_j \le n$ for $1 \le j \le k$ such that $\phi(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) = 0$. Consider set P' of k points $y_{i_1,1}, y_{i_2,2}, \ldots, y_{i_k,k}$. It implies that $\sum_{j=1}^{k} \lambda y_{i_j,j} = p$ where $\lambda = \frac{1}{k}$. Therefore, $p \in \mathsf{conv}(P')$ and $C(P, p) \le k$.

$\Longleftarrow$) Suppose there exist k pairs of integers $(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)$ where $1 \le i_t \le n$, $1 \le j_t \le k$, for $1 \le t \le k$, such that $p \in \mathsf{conv}(p_1, p_2, \ldots, p_k)$ where $p_1 = y_{i_1,j_1}, p_2 = y_{i_2,j_2}, \ldots, p_k y_{i_k,j_k}$. Therefore, there exists λ_t, for $1 \le t \le k$, such that $0 \le \lambda_t \le 1$, for $0 \le t \le k$ and $\sum_{t=1}^{k} \lambda_t = 1$ and

$$\sum_{t=1}^{k} \lambda_t p_t = p. \tag{1}$$

We claim that for every pair of integers t_1 and t_2 where $1 \le t_1 < t_2 \le k$, $j_{t_1} \ne j_{t_2}$, otherwise there exists an integer m such that $1 \le m \le k$ and $m \notin \{j_1, j_2, \ldots, j_k\}$. Then the m-th coordinate of all points $p_1, p_2, \ldots, p_k$ is zero. Then $p_m = 0$ contradicting the choice of p. Therefore, $j_1, j_2, \ldots, j_k$ is a permutation of $1, 2, \ldots, k$. By reordering points $p_1, p_2, \ldots, p_k$ we assume that $j_t = t$ for $1 \le t \le k$.

By taking mth coordinate, $1 \le m \le k$, Equation (1) implies

$$\sum_{t=1}^{k} \lambda_t p_{t,m} = \lambda_m p_{m,m} = \lambda_m = 1/k.$$

Now take $(k+1)$th coordinate in Equation (1)

$$\sum_{t=1}^{k} \lambda_t p_{t,k+1} = \sum_{t=1}^{k} \frac{1}{k} p_{t,k+1} = \sum_{t=1}^{k} \frac{1}{k} \alpha_{j_t} x_{i_t} = \frac{-\alpha_0}{k}.$$

Hence, $\alpha_0 + \sum_{t=1}^{k} \alpha_t x_{i_t} = 0$ which is the solution for k-LDT. □

4 Algorithms for ComputingCNP

We present two algorithms for ComputingCNP when $C(P, p) = 2, 3$.

4.1 $C(P, p) = 2$ in $\mathbb{R}^d$

One can easily decide in $O(n)$ time whether $C(P, p) = 1$. In this section we discuss the problem of deciding whether $C(P, p) = 2$. We assume that dimension $d \ge 2$ is a constant.

Theorem 14 *Let P be a set of n points in $\mathbb{R}^d$ and p be a point in $\mathbb{R}^d$, where $d \ge 2$ is fixed. One can decide whether $C(P, p) = 2$ and find the corresponding set P' in $O(n \log n)$ time which is optimal in the algebraic decision tree model.*

Proof. The task is to compute a subset $P' \subset P$ such that $|P'| \le 2$ and $p \in \mathsf{conv}(P')$ if it exists. We will describe an algorithm and prove the lower bound.

Algorithm. First, we decide whether $C(P, p) = 1$ in $O(n)$ time by searching p in P. Assume that $C(P, p) \ge 2$, i.e. $p_i \ne p$ for all $p_i \in P$. Compute normalized vectors $p'_i = \frac{p_i - p}{\|p_i - p\|}$. We sort points $p'_1, p'_2, \ldots, p'_n$ lexicographically and assume that they are in the lexicographic order, i.e. $p'_1 \preceq p'_2 \preceq \cdots \preceq p'_n$.

Since $C(P, p) \ne 1$, $C(P, p) = 2$ if and only if there exist two points $p_i, p_j \in P$ such that $p = ap_i + (1-a)p_j$ for some $1 \le i < j \le n$ and $0 < a < 1$ (Clearly, if $a = 0$ or $a = 1$ then $C(P, p) = 1$). This equation can be written as

$$0 = a(p_i - p) + (1 - a)(p_j - p) \tag{2}$$

$$a(p_i - p) = (a - 1)(p_j - p) \tag{3}$$

$$al_i p'_i = (a - 1) l_j p'_j, \tag{4}$$

where $l_i = \|p_i - p\|$ and $l_j = \|p_j - p\|$. Since p'_i and p'_j are unit vectors, we have $|al_i| = |(a-1)l_j|$. Note that $al_i > 0$ and $(a-1)l_j < 0$. Equation (4) implies that $p'_i = -p'_j$. Conversely, if $p'_i = -p'_j$ then $p = ap_i + (1-a)p_j$ for $a = l_j/(l_i + l_j)$.

The algorithm performs binary search of $-p'_i$ in the sorted sequence $p'_1, p'_2, \ldots, p'_n$, for each $i \in [n]$. If $-p'_i$ is found then $-p'_i = p'_j$ for some p'_j. Note that j must be not equal to i since $-p'_i = p'_i$ would mean that $p'_i = 0$ but $\|p'_i\| = 1$. The time complexity of the above algorithm is $O(n \log n)$ where $n = |P|$.

Proof of the lower bound. We now prove that the lower bound on the time complexity for the problem of deciding $C(P, p) = 2$ is $\Omega(n \log n)$. We use the 2-SUM problem: Given n numbers, do any two of them sum to zero? Chan, Gasarch and Kruskal [10] proved that solving 2-SUM in algebraic decision tree model takes $\Omega(n \log n)$ time. Let $X = \{x_1, x_2, \ldots, x_n\}$ be set of integers in an instance of 2-SUM.

We construct a set P of n points in $\mathbb{R}^2$. If $d \geq 3$, we can use a 2-dimensional plane in $\mathbb{R}^d$ for the points in P. Assign point $p = (0, 0)$. Consider the map $\mu : \mathbb{R} \to \mathbb{R}^2$ defined as $\mu(x) := (\mathsf{sgn}(x), x)$ where

$$\mathsf{sgn}(x) = \begin{cases} x/|x| & \text{if } x \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Find set $P = \mu(X)$ in $O(n)$ time, i.e. $P = \{p_i \mid p_i = \mu(x_i), i \in [n]\}$. To show that the reduction is correct, we prove the following claim. There are distinct integers i, j such that $x_i + x_j = 0$ if and only if there are two points $p_i, p_j \in P$ for which $p \in \mathsf{conv}(\{p_i, p_j\})$.

Suppose that $x_i + x_j = 0$ for some integers i, j. Then either $x_i = x_j = 0$ or $x_i x_j < 0$. If $x_i = x_j = 0$ then $p_i = (0, 0) \in P$, so $p \in \mathsf{conv}(\{p_i\})$. If $x_i x_j < 0$, then we can assume $x_i < 0 < x_j$. Then $0 \in \mathsf{conv}(\{p_i, p_j\})$ since $p_i = -p_j$.

Now suppose that there exists a subset $P' \subset P$ such that $|P'| = 2$ and $p \in \mathsf{conv}(P')$. If $p \in P'$, then $0 \in X$. If $p \notin P'$ and $P' = \{p_i, p_j\}$, then x_i and x_j have opposite signs (since $p_x = 0$). Then the convex combination $p = ap_i + (1-a)p_j$ must have $a = 1 - a$ (using x-coordinates in $p = ap_i + (1-a)p_j$, $0 = a \cdot \mathsf{sgn}(x_i) + (1-a) \cdot \mathsf{sgn}(x_j)$). Then $a = 1/2$ and $x_i = -x_j$ (using y-coordinates in $p = ap_i + (1-a)p_j$). □

4.2 $C(P, p) = 3$ in $\mathbb{R}^3$

Theorem 15 *Let P be a set of n points in $\mathbb{R}^3$ and p be a point in $\mathbb{R}^3$. One can decide whether $C(P, p) = 3$ and find the corresponding set P' in $O(n^2 \log n)$ time.*

Let $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^3$. We denote by $\alpha_{i,j}$ the angle between vectors $\overrightarrow{pp_i}$ and $\overrightarrow{pp_j}$, i.e. $\cos \alpha_{i,j} = \frac{\overrightarrow{pp_i} \cdot \overrightarrow{pp_j}}{\|\overrightarrow{pp_i}\| \cdot \|\overrightarrow{pp_j}\|}$ and $0 \leq \alpha_{i,j} \leq \pi$. Let k be an integer in $\{1, 2, \ldots, n\}$. Consider the plane π_k passing through point p with $\overrightarrow{pp_k}$ is its normal vector. Let q_i be the projection of p_i on π_k, see Fig. 3. We apply the following algorithm.

Input: $p \in \mathbb{R}^3$ and $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^3$ such that $p \in \mathsf{conv}(P)$.

Output: Decide if $C(P, p) = 3$. If $C(P, p) = 3$, compute a subset $S \subset P$ such that $|S| = 3$ and $p \in \mathsf{conv}(S)$.

1. Check if $C(P, p) = 1$ or $C(P, p) = 2$ from Section 4.1. Stop if $C(P, p) \leq 2$.
2. For each point $p_k \in P$ do the following:
3. Compute plane π_k (it can be computed since $p \neq p_k$; otherwise $C(P, p) = 1$). Compute points q_i for all $i \in \{1, 2, \ldots, n\}, i \neq k$, see Fig. 3.
4. Compute set P_k as follows. Initialize $P_k = P \setminus \{p_k\}$, then prune P_k by repeating the following step.

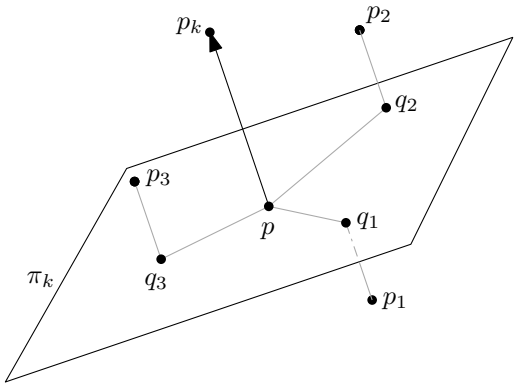


Figure 3: Plane π_k is orthogonal to vector $\overrightarrow{pp_k}$. Point $q_i, i = 1, 2, 3$ is the projection of point p_i onto the plane π_k.

5. **The pruning step**. Remove a point p_i from P_k, if there exists another point p_j in P_k such that
   (a) Vectors $\overrightarrow{pq_i}$ and $\overrightarrow{pq_j}$ have same direction and
   (b) $\alpha_{i,k} < \alpha_{j,k}$ (if $\alpha_{i,k} = \alpha_{j,k}$ remove either p_i or p_j from P_k).
6. Compute $Q_k = \{q_i \mid p_i \in P_k\}$. For each point $q_i \in Q_k$, use the binary search for $-q_i$ in Q_k as in the algorithm from Section 4.1. Suppose $C(Q_k, p) = 2$ and a set $Q' = \{q_i, q_j\}$ is found such that $p \in \mathsf{conv}(q_i, q_j)$. Check if $p \in \mathsf{conv}(\{p_i, p_j, p_k\})$ in $O(1)$ time. If $p \in \mathsf{conv}(\{p_i, p_j, p_k\})$ then output the solution $P' = \{p_i, p_j, p_k\}$. If $p \notin \mathsf{conv}(\{p_i, p_j, p_k\})$, check next point q_i in the loop.
7. If a solution is not found in Step 6, then there is no solution for $C(P, p) = 3$, so $C(P, p) = 4$.

First, we justify the pruning step.

Lemma 16 *Suppose $p \in \mathsf{conv}(\{p_i, p_j, p_k\})$ for some points $p_i, p_j, p_k \in \mathbb{R}^3$. If p_i or p_j is removed in the pruning step for p_k then there exist $p_{i'}, p_{j'} \in P_k$ such that $p \in \mathsf{conv}(\{p_{i'}, p_{j'}, p_k\})$.*

Proof. It suffices to prove the lemma if only one point from $\{p_i, p_j\}$ is removed in the pruning step. If both of them are removed, the argument can be used twice, see Fig. 4(b) for an example.

Suppose that point p_i is removed in the pruning step for p_k. Then there exists another point $p_{i'}$ in P such that

1. Vectors $\overrightarrow{pq_i}$ and $\overrightarrow{pq_{i'}}$ have same direction and
2. $\alpha_{i,k} \leq \alpha_{i',k}$

Then points p, p_i, p_j, p_k are coplanar. Without loss of generality we can assume $p_i, p_{i'}, p_j, p_k \in \mathbb{R}^2$, p_k is on

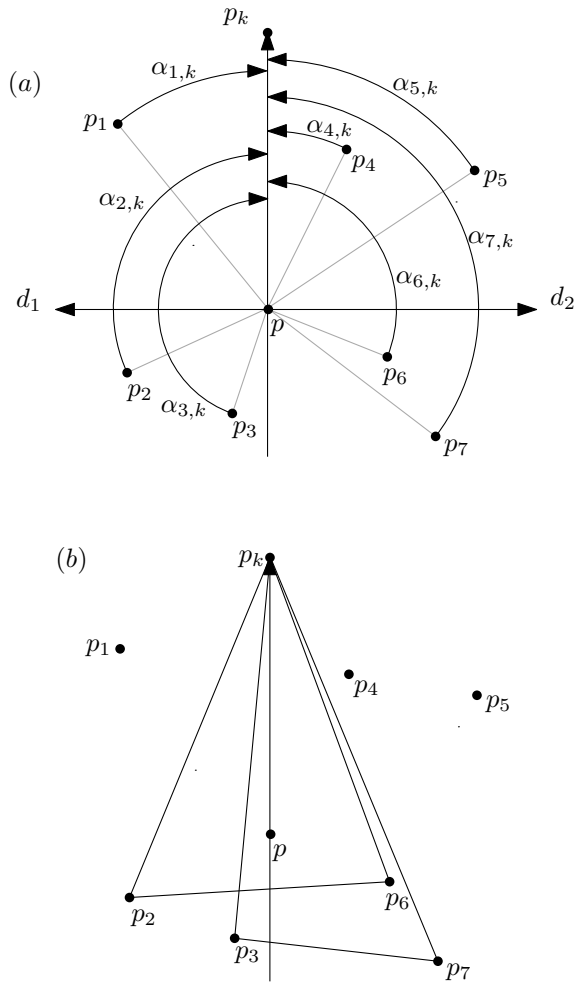


Figure 4: (a) The pruning step. Vectors $\overrightarrow{pq_i}, i = 1, 2, 3$ have the same direction d_1. Points p_1 and p_2 will be pruned based on α-angles. Vectors $\overrightarrow{pq_i}, i = 4, 5, 6, 7$ have the same direction $d_2 = -d_1$. Points $p_i, i = 4, 5, 6$ will be pruned based on α-angles. (b) Point $p \in \mathsf{conv}(\{p_k, p_2, p_6\})$ before the pruning for p_k and it is in $\mathsf{conv}(\{p_k, p_3, p_7\})$ after the pruning for p_k.

the y-axis and p is at the origin. We can assume that $x(p_i), x(p_{i'}) > 0$ and $x(p_j) < 0$. The necessary and sufficient condition for $p \in \mathsf{conv}(\{p_i, p_j, p_k\})$ is $\alpha_{i,k} + \alpha_{j,k} > \pi$. Since $\alpha_{i,k} \leq \alpha_{i',k}$, we have $\alpha_{i',k} + \alpha_{j,k} > \pi$. Therefore $p \in \mathsf{conv}(\{p_{i'}, p_j, p_k\})$ and the lemma follows. □

Time Complexity. Plane π_k is computed in $O(1)$ time. Projection of P onto π_k takes $O(n)$ time. The pruning step can be done in $O(n \log n)$ time by maintaining the sorted order of points q_i by the direction. Finally, Step 6 takes $O(n \log n)$ time since binary search takes $O(\log n)$ time and it is done for every point in Q_k. Therefore, the processing of p_k takes $O(n \log n)$ time and the total time is $O(n^2 \log n)$.

5 Hardness and Algorithms for ComputingCCNP

In this section we show that our results for ComputingCNP can be extended directly to ComputingCCNP.

5.1 Hardness

We show that DecidingCCNP (i.e. deciding if $CC(P, p) \leq k$), the decision version of ComputingCCNP, is NP-hard. There is a natural reduction from DecidingCNP problem to DecidingCCNP problem. Consider an instance of DecidingCNP, i.e. a set of points P in $\mathbb{R}^d$, a point $p \in \mathsf{conv}(P)$ and an integer k. We construct an instance of DecidingCCNP by taking $d + 1$ copies of P, the color classes $\mathcal{P} = \{P_1, \ldots, P_{d+1}\}$ and by using the same point p and integer k. Clearly, this reduction can be computed in polynomial time. It remains to prove that $C(P, p) \leq k$ if and only if $CC(\mathcal{P}, p) \leq k$. If $C(P, p) \leq k$ then there exists a subset $P' = \{p_1, p_2, \ldots, p_k\}$ of P such that $p \in \mathsf{conv}(P')$. Then $CC(\mathcal{P}, p) \leq k$ by selecting p_i from color set P_i (i.e. P' is a rainbow for $\mathcal{P}$). Similarly, $CC(\mathcal{P}, p) \leq k$ implies $C(P, p) \leq k$. Therefore DecidingCCNP is NP-hard.

Similarly, one can prove that DecidingCCNP is k-LDT-hard if dimension d is fixed (we omit details due to lack of space).

5.2 $CC(\mathcal{P}, p) = 2$ in $\mathbb{R}^d$

We show that the algorithm from Section 4.1 can be modified for deciding if $CC(\mathcal{P}, p) = 2$ in $\mathbb{R}^d$ and computing the corresponding rainbow. In this problem, we have $d + 1$ color classes, and they can be processed as follows. We normalize the vectors (of all colors) again, but this time there could be equal normal vectors of different colors. We store one vector for them and the list of their colors. Then the binary search is modified to select a vector of different color from the list.

The time complexity of this algorithm is $O(n \log n)$ where $n = \sum_{i=1}^{d+1} |P_i|$.

5.3 $CC(P, p) = 3$ in $\mathbb{R}^3$

We briefly (due to lack of space) show that the algorithm from Section 4.2 can be modified for deciding if $CC(\mathcal{P}, p) = 3$ in $\mathbb{R}^3$ and computing the corresponding rainbow. In step 2, we select p_k from $\cup P_i$. In step 3, we use the same colors for projected points. In the pruning step, if there are more than two points q_i and q_j with distinct colors with the same direction of $\overrightarrow{pq_i}$ and $\overrightarrow{pq_j}$, we store the two with the largest α-angles. In steps 6, we apply the algorithm for deciding $CC(\mathcal{P}, p) = 2$ instead of deciding $C(P, p) = 2$. The total time complexity of the algorithm is $O(n^2 \log n)$ where $n = \sum_{i=1}^{d+1} |P_i|$.

References

[1] I. Bárány. A generalization of Carathéodory's theorem. *Discrete Mathematics*, 40(2-3):141–152, 1982.

[2] I. Bárány and R. Karasev. Notes about the Carathéodory number. *Discrete & Computational Geometry*, 48(3):783–792, 2012.

[3] I. Bárány and S. Onn. Colourful linear programming and its relatives. *Mathematics of Operations Research*, 22(3):550–567, 1997.

[4] R. M. Barbosa, E. M. Coelho, M. C. Dourado, D. Rautenbach, and J. L. Szwarcfiter. On the Carathéodory number for the convexity of paths of order three. *SIAM Journal on Discrete Mathematics*, 26(3):929–939, 2012.

[5] G. Barequet and S. Har-Peled. Polygon containment and translational in-hausdorff-distance between segment sets are 3sum-hard. *International Journal of Computational Geometry & Applications*, 11(04):465–474, 2001.

[6] S. Barman. Approximating Nash equilibria and dense bipartite subgraphs via an approximate version of Caratheodory's theorem. In *Proc. 47th ACM symposium on Theory of computing*, pages 361–369. ACM, 2015.

[7] V. Bui and R. Karasev. On the Carathéodory number for strong convexity. *arXiv preprint arXiv:1806.10937*, 2018.

[8] C. Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907.

[9] J. Cardinal and A. Ooms. Sparse regression via range counting. In *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22-24, 2020, Tórshavn, Faroe Islands*, pages 20:1–20:17, 2020.

[10] A. C. Chan, W. I. Gasarch, and C. P. Kruskal. Refined upper and lower bounds for 2-SUM. `https://www.researchgate.net/publication/228515076_Refined_Upper_and_Lower_Bounds_for_2-SUM`.

[11] E. M. Coelho, M. C. Dourado, D. Rautenbach, and J. L. Szwarcfiter. The Carathéodory number of the P3 convexity of chordal graphs. *Discrete Applied Mathematics*, 172:104–108, 2014.

[12] M. C. Dourado, D. Rautenbach, V. F. Dos Santos, P. M. Schäfer, and J. L. Szwarcfiter. On the Carathéodory number of interval and graph convexities. *Theoretical Computer Science*, 510:127–135, 2013.

[13] J. Eckhoff. Helly, Radon, and Carathéodory type theorems. In *Handbook of convex geometry*, pages 389–448. Elsevier, 1993.

[14] J. Erickson. Lower bounds for linear satisfiability problems. In *SODA*, pages 388–395, 1995.

[15] W. Fenchel. Über krümmung und windung geschlossener raumkurven. *Mathematische Annalen*, 101(1):238–252, 1929.

[16] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[17] O. Hanner and H. Rådström. A generalization of a theorem of Fenchel. *Proceedings of the American Mathematical Society*, 2(4):589–593, 1951.

[18] S. Har-Peled, P. Indyk, and S. Mahabadi. Approximate sparse linear regression. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 77:1–77:14, 2018.

[19] M. Ito and B. F. Lourenço. A bound on the Carathéodory number. *Linear Algebra and its Applications*, 532:347–363, 2017.

[20] D. S. Johnson and M. R. Garey. *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.

[21] R. M. Karp. Reducibility among combinatorial problems. In *Proc. of a Symposium on the Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[22] D. Kay and E. W. Womble. Axiomatic convexity theory and relationships between the Carathéodory, Helly, and Radon numbers. *Pacific Journal of Mathematics*, 38(2):471–485, 1971.

[23] F. Meunier, W. Mulzer, P. Sarrabezolles, and Y. Stein. The rainbow at the end of the line—a ppad formulation of the colorful carathéodory theorem with applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1342–1351. SIAM, 2017.

[24] W. Mulzer and Y. Stein. Computational aspects of the colorful Carathéodory theorem. *Discrete & Computational Geometry*, 60(3):720–755, 2018.

[25] S. Naldi. Nonnegative polynomials and their Carathéodory number. *Discrete & Computational Geometry*, 51(3):559–568, 2014.

[26] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234, 1995.

[27] G. Sierksma. Carathéodory and Helly-numbers of convex-product-structures. *Pacific Journal of Mathematics*, 61(1):275–282, 1975.

[28] G. M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, 1994.

Characterization and Computation of Feasible Trajectories for an Articulated Probe with a Variable-Length End Segment

Ovidiu Daescu* Ka Yaw Teo*

Abstract

We consider an extension of the articulated probe trajectory planning problem introduced in [11], where the length r of the end segment can be customized. We prove that, for n line segment obstacles, the smallest length r for which there exists a feasible probe trajectory can be found in $O(n^{2+\epsilon})$ time using $O(n^{2+\epsilon})$ space, for any constant $\epsilon > 0$. Furthermore, we prove that all values r for which a feasible probe trajectory exists form $O(n^2)$ intervals, and can be computed in $O(n^{5/2})$ time using $O(n^{2+\epsilon})$ space. We also show that, for a given r, the feasible trajectory space of the articulated probe can be characterized by a simple arrangement of complexity $O(n^2)$, which can be constructed in $O(n^2)$ time. To obtain our solutions, we design efficient data structures for a number of interesting variants of geometric intersection and emptiness query problems.

1 Introduction

The *articulated probe trajectory planning problem* was introduced in [11] with the following setup. We are given a two-dimensional workspace containing a set P of simple polygonal obstacles with a total of n vertices, and a target point t in the free space, all enclosed by a circle S of radius R centered at t. An articulated probe is modeled in $\Re^2$ as two line segments, ab and bc, connected at point b. The length of ab is greater than or equal to R, whereas bc is of some small length $r \in (0, R]$. The probe is initially located outside S, assuming an *unarticulated* configuration, in which ab and bc are collinear, and $b \in ac$. A *feasible probe trajectory* consists of an initial insertion (sliding) of straight line segment abc into S, possibly followed by a rotation of bc around b up to $\pi/2$ radians in either direction, such that c coincides with t, while avoiding the obstacles in the process. If a rotation is performed, then we have an *articulated final* configuration of the probe. The goal is to determine if a feasible probe trajectory exists and, if so, to report one such trajectory.

It has been argued in [11] that the polygonal obstacles can be treated by considering only their bounding line segments. Thus, for simplicity, assume that P consists of n non-crossing line segment obstacles (Figure 1). We further assume that S is not visible from t, since otherwise the problem reduces to computing visibility from t to infinity, which takes $O(n \log n)$ time [11]. Thus, in order to reach t, the probe has to rotate bc around b.

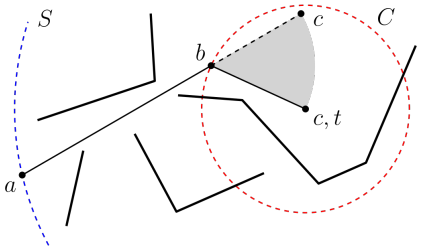


Figure 1: In order to reach the target point t, a straight insertion of line segment abc may be followed by a rotation of bc from its intermediate position (dashed line) to its final position (solid line).

After inserting segment abc, point a is located on or outside S. Let C be the circle of radius r centered at t. Observe that, since bc may only rotate as far as $\pi/2$ radians in either direction after the initial insertion of segment abc, ab intersects C only once and at b (i.e., $b \in C$). When bc rotates around b, the area swept by bc is a sector of a circle of radius r centered at b. For conciseness, the center of the circle on which a circular sector is based is called the *center of the circular sector*.

In this paper, we develop efficient algorithms for computing i) the minimum value $r > 0$ for which a feasible articulated trajectory exists, including reporting at least one such trajectory, ii) all values $r > 0$ for which a feasible articulated trajectory exists (i.e., *feasible domain* of r), and iii) the feasible trajectory space (i.e., set of all feasible trajectories) for a given value r.

Related work. The two-dimensional articulated probe trajectory planning problem (with a constant length r) was originally introduced by Teo, Daescu, and Fox [11], who presented a geometric-combinatorial algorithm for computing so-called *extremal* feasible probe trajectories in $O(n^2 \log n)$ time using $O(n \log n)$ space. In an extremal probe trajectory, one or two obstacle endpoints always lie tangent to the probe. The solution approach proposed in [11] can be extended to the case of polyg-

*Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA. {ovidiu.daescu, ka.teo}@utdallas.edu

onal obstacles. For h polygonal obstacles with a total of n vertices, an extremal feasible probe trajectory can be determined in $O(n^2 + h^2 \log h)$ time using $O(n \log n)$ space. When a clearance δ from the polygonal obstacles is required, a feasible probe trajectory can be obtained in $O(n^2 + h^2 \log h)$ time using $O(n^2)$ space.

In addition, Daescu and Teo [4] developed an algorithm for solving the articulated probe trajectory planning problem in three dimensions for a given r. It was shown that a feasible probe trajectory among n triangular obstacles can be found in $O(n^{4+\epsilon})$ time using $O(n^{4+\epsilon})$ space, for any constant $\epsilon > 0$.

Motivation. Besides its general relevance in robotics, the proposed problem arises specifically in some medical applications. In minimally invasive surgeries, a rigid needle-like instrument is typically inserted through a small incision to reach a given target, after which it may perform operations such as tissue resection and biopsy. Some newer designs allow for a joint to be incorporated for moving the acting end (tip); after inserting the instrument in a straight path, the surgeon may rotate the tip around the joint to reach the target [9].

Due to the rapid advances in three dimensional printing techniques, such robotic probes can even be customized for a given patient [3]. Rather than using a one-size-fits-all instrument, based on the patient-specific requirement and constraints, a robotic probe with a tailored-sized tip can be customarily built on-demand using three dimensional printing.

Despite its importance and relevance, as well as its rich combinatorial and geometric properties, only a handful of results have been reported [4, 11] for this trajectory planning problem.

Results and contributions. Recall our assumption that there is no feasible unarticulated probe trajectory (i.e., t cannot see to infinity). We begin in Section 2 by addressing our first problem of interest:

Problem 1 *Find the minimum length $r > 0$ of segment bc such that a feasible articulated probe trajectory exists, if any, and report (at least) one such trajectory.*

For brevity, a feasible articulated trajectory with the minimum length r is referred to as a feasible min-r articulated trajectory. Our approach to solving Problem 1 is as follows: i) We show that a feasible min-r articulated trajectory, if one exists, can always be perturbed, while remaining feasible, into one of a finite number of "extremal" feasible trajectories, which can be enumerated using an algebraic-geometric method (see Lemma 1 for a detailed definition of the extremal trajectories). This leads to a simple $O(n^3 \log n)$ time, $O(n^{2+\epsilon})$ space algorithm, for any constant $\epsilon > 0$, based on enumerating and verifying the extremal trajectories for feasibility (see the full version of the paper for details). ii) We then derive an $O(n^{2+\epsilon})$ time and space algorithm by partially waiving the notion of computing and checking the extremal trajectories for feasibility. Specifically, the algorithm searches for a feasible min-r articulated trajectory, if any, by performing a finite sequence of perturbations and feasibility tests on certain $O(n^2)$ extremal trajectories (whose segment ab is tangent to two obstacle endpoints). With a proper algorithmic extension to our process of finding the minimum feasible r, we can compute, in $O(n^{5/2})$ time using $O(n^{2+\epsilon})$ space, the set of r-intervals for which feasible articulated trajectories exist, together with an implicit representation of feasible solutions for those values of r.

In the process of deriving our solution to Problem 1, we encounter and solve a number of fundamental problems (or their special cases) that could be of theoretical interest in computational geometry. For instance, we provide an efficient data structure with logarithmic query time for solving a special instance of the circular sector emptiness query problem (i.e., for a query circular sector with a fixed arc endpoint t).

In Section 3, we address our second problem:

Problem 2 *For a given length r of segment bc, compute the feasible trajectory space (i.e., set of all feasible trajectories) of the articulated probe.*

We describe a geometric combinatorial approach for characterizing and computing the feasible trajectory space of the articulated probe. The feasible configuration space has a worst-case complexity of $O(n^2)$ and can be described by an arrangement of simple curves. Using topological sweep [2], the arrangement can be constructed in $O(n \log n + k)$ time using $O(n + k)$ working storage, where $k = O(n^2)$ is the number of vertices of the arrangement. By simply traversing the cells of the arrangement, we can find a feasible probe trajectory in $O(n^2)$ time – a logarithmic factor improvement compared to the algorithm in [11].

2 Computing feasible min-r articulated trajectories

Recall that, for a given r, C is the circle of radius r centered at t. Using the rationale of [11, Lemma 2.1], we can immediately claim the following observation.

Observation 1 *Given a feasible min-r articulated trajectory, there exists an **extremal** feasible min-r articulated trajectory such that the probe assumes an articulated final configuration that passes through an obstacle endpoint outside C and another obstacle endpoint inside or outside C.*

We will later show in Lemma 1 that an extremal feasible min-r articulated trajectory is always tangent to two obstacle endpoints *outside* C.

For ease of discussion, unless noted otherwise, we use bc and bt to denote line segment bc of the probe in its intermediate (right after the initial insertion of segment abc) and final configurations, respectively. Let $\angle cbt$ be the angle of rotation of segment bc to reach t, and let σ_{bct} be the circular sector swept by segment bc in order to reach t. Let γ_{ct} denote the circular arc of σ_{bct}. Let V denote the set of endpoints of the line segments of P.

Lemma 1 *Given a feasible min-r articulated trajectory, there exists an* ***extremal*** *feasible min-r articulated trajectory such that, in its final configuration, ab passes through two obstacle endpoints and at least one of the following holds: I) $\angle cbt = \pi/2$ radians, II) bc intersects an obstacle line segment at c, III) γ_{ct} intersects an obstacle endpoint or is tangent to an obstacle line segment, IV) one of the obstacle endpoints intersected by ab coincides with b, and $\angle cbt \leq \pi/2$ radians, or V) bt passes through an obstacle endpoint.*

The full proof of Lemma 1 is given in Appendix A.

Solution approach. We begin by emphasizing that, as stated in Lemma 1, an extremal feasible min-r articulated trajectory passes through two obstacle endpoints, neither of which is inside C.

Consider the following solution approach. For each point $v \in V$, compute the set R_v of rays with the following properties: i) Each ray originates at v and passes through a point $u \in V \setminus \{v\}$. ii) Segment vb_0 does not intersect any line segment of P, where b_0 is the point of tangency between the supporting line of the ray and the circle C centered at t (Figure 2A). iii) If the ray passes through b_0, then the *reversal* of the ray does not intersect any line segment of P; otherwise, the ray itself does not intersect any line segment of P. R_v can be obtained in $O(n \log n)$ time by computing the visibility polygon from v [1, 7, 10]. Since $|V| = O(n)$, the worst-case running time for finding the set of rays $R = \cup_{v \in V} R_v$ is $O(n^2 \log n)$.

Note that each ray of R is associated with a trajectory T that has an obstacle-free segment ab passing through two obstacle endpoints. Without loss of generality, assume that ab of T passes through a pair of obstacle endpoints $u, v \in V$, where $u \neq v$, in the way depicted in Figure 2A. Assume that bc of T rotates clockwise to reach t (the other case is symmetrical). Let b_0 be the position of b when $\angle cbt = \pi/2$ radians, and c_0 be the position of c when $b = b_0$. In order to find a feasible min-r articulated trajectory, we perform the following sequence of steps.

A1. Check if the articulated trajectory T with $\angle cbt = \pi/2$ radians is feasible. Specifically, check if the quarter circular sector bounded by b_0c_0, b_0t, and circular arc γ_{c_0t} (centered at b_0 and emanating counter-clockwise from t to c_0) is free of obstacles (Figure 2A). If it is, then T is a feasible min-r articulated trajectory whose ab passes through u and v. Otherwise, proceed with step A2.

A2. Check if b_0t is intersected by any obstacle (Figure 2A). If it is, then a feasible min-r articulated trajectory whose ab passes through u and v does not exist. Otherwise, proceed with steps A3 and A4.

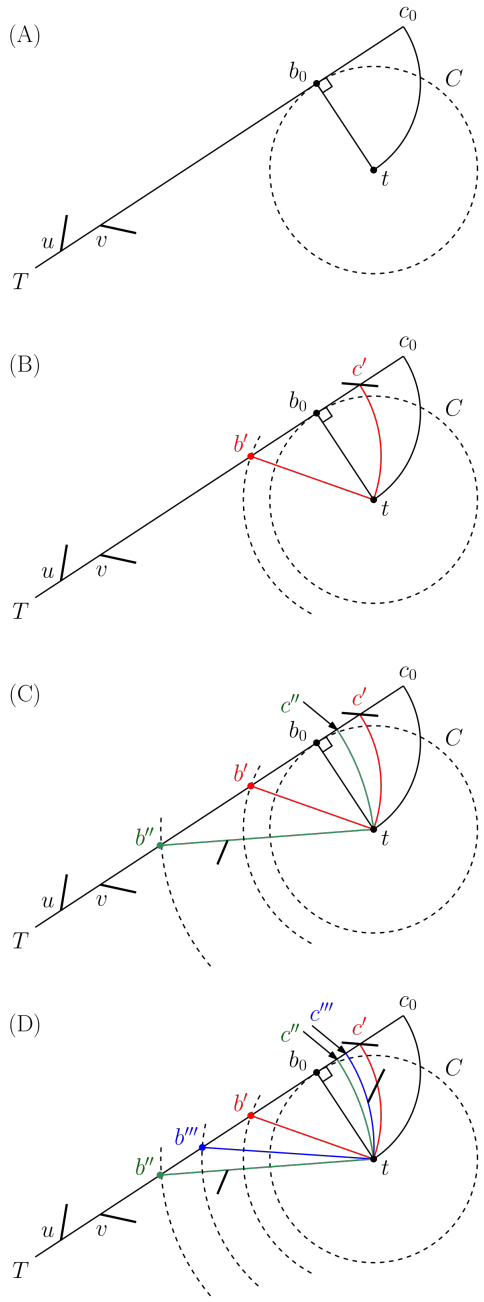


Figure 2: Illustrations of steps (A) A1 and A2, (B) A3 and A4, (C) A5 and A6, and (D) A7.

Table 1: Summary of query data structures used in steps A1-A7. The size, preprocessing time, and query time of a data structure are denoted by $S(n)$, $P(n)$, and $Q(n)$, respectively.

Step	**Query data structure**	$\boldsymbol{S(n)}$	$\boldsymbol{P(n)}$	$\boldsymbol{Q(n)}$
A1, A6	Circular sector emptiness queries	$O(n^{2+\epsilon})$	$O(n^{2+\epsilon})$	$O(\log n)$
A2, A4	Radius intersection queries	$O(n)$	$O(n \log n)$	$O(\log n)$
A3	Ray shooting queries [8]	$O(n^2)$	$O(n^2)$	$O(\log n)$
A5	Radius shooting queries	$O(n^2/\log^2 n)$	$O(n^2/\log^2 n)$	$O(\log^2 n)$
A7	Arc shooting queries	$O(n^{2+\epsilon})$	$O(n^{2+\epsilon})$	$O(\log n)$

A3. Find the closest point $c' \in b_0c_0$ to c_0 such that b_0c' does not intersect any obstacle (Figure 2B). Compute the center b' of the circular arc $\gamma_{c't}$ emanating counter-clockwise from t to c', where $b' \in vb_0$.

A4. Check if $b't$ is intersected by any obstacle (Figure 2B). If it is, then a feasible min-r articulated trajectory whose ab passes through u and v does not exist. Otherwise, proceed with steps A5 and A6.

A5. Find the closest point $b'' \in vb'$ to b' such that $b''t$ intersects an obstacle endpoint (Figure 2C). Compute the corresponding point c'' (i.e., the intersection between b_0c' and the circle of radius $|b''t|$ centered at b''). Note that the triangle bounded by $b't$, $b''t$, and $b'b''$ is free of obstacles.

A6. Check if the "sector" bounded by $b'c''$, $b't$, and circular arc $\gamma_{c''t}$ (centered at b'' and emanating counter-clockwise from t to c'') intersects any obstacle (Figure 2C). Note that it is equivalent to checking if the circular sector bounded by $b''c''$, $b''t$, and $\gamma_{c''t}$ intersects any obstacle. If it does, then a feasible min-r articulated trajectory whose ab passes through u and v does not exist. Otherwise, proceed with step A7.

A7. At this point, observe that the articulated trajectory with the intermediate configuration represented by $ab''c''$ is feasible. Find the closest point $b''' \in b'b''$ to b' such that circular arc $\gamma_{c'''t}$ (centered at b''' and emanating counter-clockwise from t to c''') intersects an obstacle endpoint or is tangent to an obstacle line segment (Figure 2D). Note that the "sector" bounded by $b'c'''$, $b't$, and circular arc $\gamma_{c'''t}$ is free of obstacles. The articulated trajectory with the intermediate configuration indicated by $ab'''c'''$ is a feasible min-r articulated trajectory whose ab passes through u and v.

By simply performing an $O(n)$-check (i.e., check against each of the $O(n)$ obstacles) in each of the steps above, we can obtain an $O(n^3)$-time "brute-force" method to find a feasible min-r articulated trajectory, if one exists. Alternatively, we can address these steps using efficient data structures, which require geometric constructs such as lower envelopes and half-space decomposition schemes. Refer to Table 1 for a summary of the query data structures, whose details are deferred to the full version of the paper. $O(n^2)$ queries are to be processed in the worst case, resulting in a total query time bounded by $O(n^2 \log^2 n)$. Since the preprocessing time of the query data structures is dominant overall, we have the following final result.

Theorem 2 *A feasible min-r articulated probe trajectory, if one exists, can be determined in $O(n^{2+\epsilon})$ time using $O(n^{2+\epsilon})$ space, for any constant $\epsilon > 0$.*

The solution approach just described can be extended to find all feasible values of r. The details of the algorithmic extension will be presented in the full publication, and the corresponding result is summarized in the following theorem.

Theorem 3 *All values of r for which at least one feasible trajectory exists can be determined in $O(n^{5/2})$ time using $O(n^{2+\epsilon})$ space, for any constant $\epsilon > 0$.*

3 Characterizing feasible trajectory space

In this section, we describe our solution to Problem 2. We begin by explicitly characterizing the following for a given length r: i) the final configuration space, ii) the *forbidden* final configuration space, and iii) the *infeasible* final configuration space.

3.1 Final configuration space

In a final configuration of the articulated probe, point a can be assumed to be on S, and point b lies on the circle C of radius r centered at t (Figure 1). Let θ_S and θ_C be the angles of line segments ta and tb measured counter-clockwise from the x-axis, where $\theta_S, \theta_C \in [0, 2\pi)$. Since bc may rotate around b as far as $\pi/2$ radians in either direction, for any given θ_S, we have $\theta_C \in [\theta_S - \cos^{-1} r/R, \theta_S + \cos^{-1} r/R]$. We call this the *unforbidden* range of θ_C. A final configuration of the articulated probe can be specified by (θ_S, θ_C), depending on the locations of points a and b on circles S and C,

respectively (Figure 3). The final configuration space Σ_{fin} of the probe can be computed in $O(1)$ time.

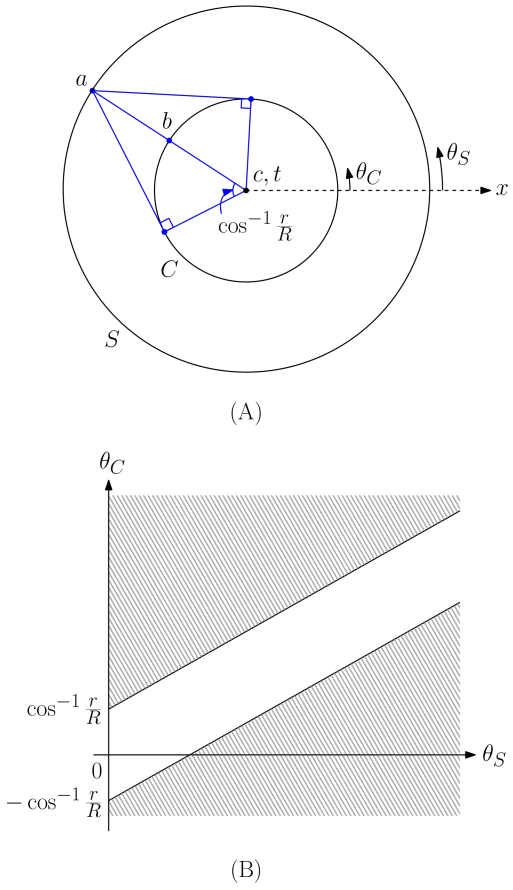


Figure 3: Final configurations of the articulated probe. (A) Each value of θ_S is associated with an unforbidden range of θ_C spanning from $\theta_S - \cos^{-1} r/R$ to $\theta_S + \cos^{-1} r/R$. (B) The unshaded region of the (θ_S, θ_C)-plot represents the unforbidden final configuration space when S is obstacle-free.

3.2 Forbidden final configuration space

A final configuration is called *forbidden* if the final configuration (represented by ab and bt) intersects one or more of the obstacle line segments. Let s be an obstacle line segment of P. We have two different cases, depending on whether s is located 1) outside or 2) inside C.

Case 1. Obstacle line segment s outside C. Let angles θ_i, where $i = 1, \ldots, 6$, be defined in the manner depicted in Figure 4A. Briefly, each θ_i corresponds to an angle θ_S at which point a tangent line i) between C and s or ii) from t to s, intersects S. As θ_S increases from θ_1 to θ_3, the upper bound of the unforbidden range of θ_C decreases as a continuous function of θ_S. Similarly, when θ_S varies from θ_4 to θ_6, the lower bound of the unforbidden range of θ_C decreases as a continuous function of θ_S. For $\theta_3 \leq \theta_S \leq \theta_4$, there exists no unforbidden final configuration at any θ_C (Figure 4B). For conciseness, the upper (resp. lower) bound of the unforbidden range of θ_C is referred to as the upper (resp. lower) bound of θ_C hereafter.

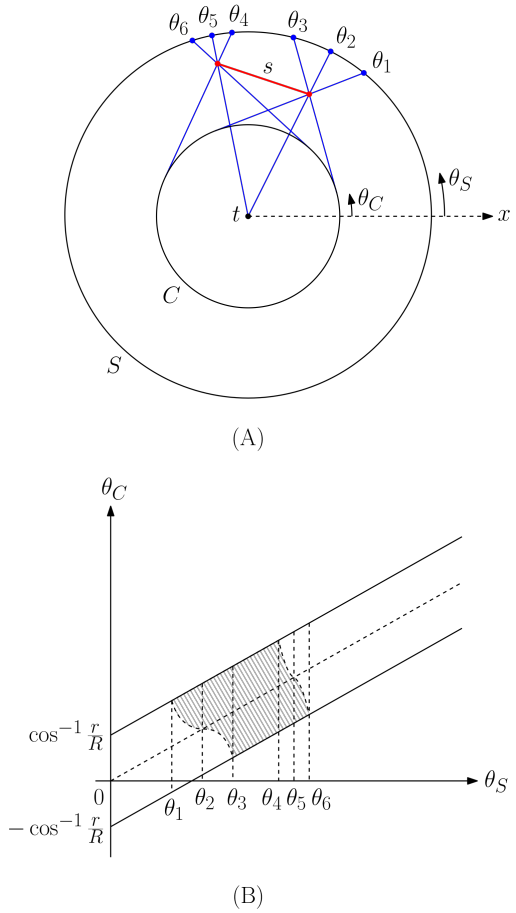


Figure 4: Forbidden final configurations due to an obstacle line segment s outside C.

Case 2. Obstacle line segment s inside C. We can similarly compute the forbidden final configuration space for an obstacle line segment s inside C. Note in Figure 5A that angles θ_i, where $i = 1, \ldots, 6$, are defined differently from case 1. For $\theta_1 \leq \theta_S \leq \theta_4$, the upper bound of θ_C is equivalent to θ_2. For $\theta_3 \leq \theta_S \leq \theta_6$, the lower bound of θ_C equals to θ_5 (Figure 5B).

We can find the forbidden final configuration space for an obstacle line segment in $O(1)$ time. Thus, for n obstacle line segments, it takes $O(n)$ time to compute the corresponding set of forbidden final configurations. The union of these configurations forms the forbidden final configuration space $\Sigma_{fin,forb}$ of the articulated probe. The free final configuration space of the articulated probe is $\Sigma_{fin,free} = \Sigma_{fin} \setminus \Sigma_{fin,forb}$.

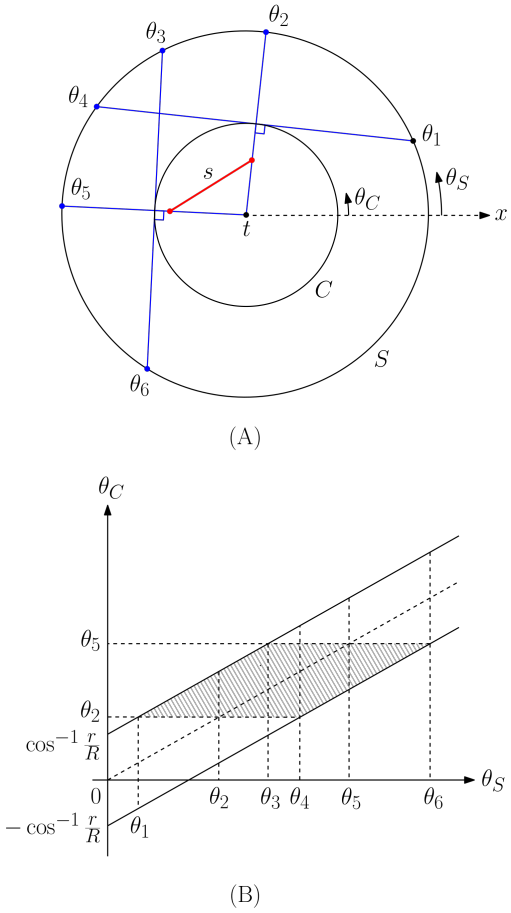


Figure 5: Forbidden final configurations due to an obstacle line segment s inside C.

3.3 Infeasible final configuration space

The feasible trajectory space of the articulated probe can be characterized as a subset of $\Sigma_{fin,free}$. A final configuration is called *infeasible* if the circular sector associated with the final configuration (i.e., the area swept by segment bc to reach t) intersects any obstacle line segment. We denote the infeasible final configuration space as $\Sigma_{fin,inf}$. The analytical details of the characterization of $\Sigma_{fin,inf}$ are presented in Appendix B. Based on the analysis, we conclude that the infeasible final configuration space associated with any obstacle line segment can be found in $O(1)$ time. As a result, it takes $O(n)$ time to determine the infeasible final configuration space for n obstacle line segments.

3.4 Complexity and construction of feasible trajectory space

The feasible trajectory space of the articulated probe is represented by $\Sigma_{fin} \setminus (\Sigma_{fin,forb} \cup \Sigma_{fin,inf})$. Three sets of lower- and upper-bound curves, denoted as σ_{fin}, $\sigma_{fin,forb}$, and $\sigma_{fin,inf}$, were obtained from characterizing the final, forbidden final, and infeasible final configuration spaces, respectively. Each of these curves is a function of θ_S – that is, $\theta_C(\theta_S)$.

As illustrated in Figure 3, σ_{fin} contains two linearly increasing curves, $\theta_C = \theta_S - \cos^{-1} r/R$ and $\theta_C = \theta_S + \cos^{-1} r/R$, which are defined over $\theta_S \in [0, 2\pi)$. Each curve in $\sigma_{fin,forb}$ is partially defined, continuous, and monotone in θ_S. Specifically, as shown in Figures 4 & 5, the curves in case 1 are monotonically decreasing with respect to θ_S, and the curves in case 2 are horizontal lines parallel to the θ_S-axis (i.e., of some constant values of θ_C). Furthermore, any two curves in case 1 can intersect at most once. Likewise, a curve in $\sigma_{fin,inf}$ is bounded and monotonically increasing with respect to θ_S (Figures 9 & 11 in Appendix B), and can intersect with another at most once.

From the observations above, it can be easily deduced that the number of intersections between any two curves in $\sigma = \sigma_{fin} \cup \sigma_{fin,forb} \cup \sigma_{fin,inf}$ is at most one. For a set σ of $O(n)$ x-monotone Jordan arcs, with at most c intersections per pair of arcs, where c is a constant, the maximum combinatorial complexity of the arrangement $A(\sigma)$ is $O(n^2)$ [6].

An incremental construction approach, as detailed in [5], can be used to construct the arrangement $A(\sigma)$ in $O(n^2\alpha(n))$ time using $O(n^2)$ space, where $\alpha(n)$ is the inverse Ackermann function. By using topological sweep [2] in computing the intersections for a collection of well-behaved curves such as those described above, the time and space complexities can be improved to $O(n \log n + k)$ and $O(n + k)$, respectively. Note that we can find a feasible probe trajectory by simply traversing the cells of the arrangement $A(\sigma)$ in $O(n^2)$ time. This implies an $O(\log n)$ improvement over the previous result reported in [11]. We thus conclude with the following theorem.

Theorem 4 *For a positive value r, the feasible trajectory space of the corresponding articulated probe can be represented as a simple arrangement of maximum combinatorial complexity $k = O(n^2)$, which can be constructed in $O(n \log n + k)$ time using $O(n + k)$ space. A feasible probe trajectory, if one exists, can be determined in $O(n^2)$ time using $O(n^2)$ space.*

4 Open questions

1) Our solution to Problem 1 relies on efficient data structures to address some rather specific geometric intersection and emptiness query problems. Can we improve upon those query data structures? 2) Do our techniques extend well to the variant in which a clearance is required from the obstacles? 3) Can we generalize our solution approaches to three dimensions?

References

[1] E. Arkin and J. Mitchell. An optimal visibility algorithm for a simple polygon with star-shaped holes. Technical report, Cornell University Operations Research and Industrial Engineering, 1987.

[2] I. J. Balaban. An optimal algorithm for finding segments intersections. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 211–219, 1995.

[3] C. Culmone, G. Smit, and P. Breedveld. Additive manufacturing of medical instruments: A state-of-the-art review. *Additive Manufacturing*, 2019.

[4] O. Daescu and K. Teo. Computing feasible trajectories for an articulated probe in three dimensions. In *31st Annual Canadian Conference on Computational Geometry*, pages 59–70, 2019.

[5] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir. Arrangements of curves in the plane – topology, combinatorics, and algorithms. *Theoretical Computer Science*, 92(2):319–336, 1992.

[6] D. Halperin and M. Sharir. Arrangements. *Handbook of Discrete and Computational Geometry*, pages 723–762, 2017.

[7] P. J. Heffernan and J. S. Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM Journal on Computing*, 24(1):184–201, 1995.

[8] M. Pocchiola. Graphics in flatland revisited. In *Scandinavian Workshop on Algorithm Theory*, pages 85–96, 1990.

[9] N. Simaan, R. M. Yasin, and L. Wang. Medical technologies and challenges of robot-assisted minimally invasive intervention and diagnostics. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:465–490, 2018.

[10] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the Second Annual Symposium on Computational Geometry*, pages 14–23. ACM, 1986.

[11] K. Teo, O. Daescu, and K. Fox. Trajectory planning for an articulated probe. *Computational Geometry*, page 101655, 2020.

Appendices

A Proof of Lemma 1

We proceed by considering the two possible scenarios implied by Observation 1.

Scenario A. A feasible min-r articulated probe trajectory exists such that ab of the trajectory passes through two obstacles endpoints $u, v \in V$, where $u \neq v$. Obviously, ab does not intersect the interior of any line segment of P. Without loss of generality, assume that segment bc of the probe is rotated clockwise around b to reach t (the other case can be handled symmetrically), and ab passes through u and v in the way depicted in Figure 6.

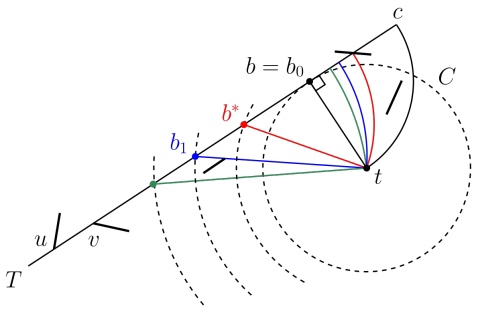


Figure 6: Finding the extremal feasible min-r articulated probe trajectory in Scenario A.

Let h_{ab} denote the supporting line of ab. Let b_0t be the perpendicular line segment dropped from t to line h_{ab}. It is easy to observe that the minimum possible value of r for an articulated trajectory is given by the length of b_0t – that is, when $b = b_0$ and $\angle cbt$ is equal to $\pi/2$ radians. Let T denote the corresponding trajectory. If T is free of obstacles, then T is a feasible min-r articulated trajectory (case I of the lemma).

Otherwise, the minimum feasible value of r is attained at some point b^* on line segment vb_0, where b^* is the closest point to b_0 on vb_0 for which the corresponding articulated trajectory is feasible. In order to find b^*, we increase r by moving b away from b_0 on vb_0 until the trajectory becomes feasible. Observe that, if bt intersects an obstacle line segment at any given time during the process of increasing r, then the trajectory would never become feasible thereafter (illustrated by the blue and green trajectories in Figure 6).

The observations above imply that, if $b = b_0$ is not feasible, then bc or γ_{ct} of T must be intersected by an obstacle line segment, or σ_{bct} of T must contain an obstacle line segment. By moving b away from b_0 on vb_0, we may rid the trajectory of obstacle line segments that intersect bc, γ_{ct}, or are contained within σ_{bct}. Suppose that we increase r until either bt becomes tangent to an obstacle line segment or b reaches v. Let b_1 denote the final position of b. Observe that b^* must lie somewhere between b_0 and b_1. In fact, as we increase r, $b = b^*$ when bc intersects an obstacle line segment at c, or γ_{ct} intersects an obstacle endpoint or is tangent to an obstacle line segment (cases II and III of the lemma).

Remark. Let r_{b_0}, r_{b^*}, and r_{b_1} be the lengths of bc when $b = b_0$, $b = b^*$, and $b = b_1$, respectively, where $r_{b_0} \leq r_{b^*} \leq r_{b_1}$. Observe that $[r_{b^*}, r_{b_1}]$ is a feasible contiguous subset of $[r_{b_0}, r_{b_1}]$. Indeed, based on the observations made thus far, it is easy to figure that, in Scenario A, there exists at most one contiguous feasible subset of $[r_{b_0}, r_{b_1}]$.

Scenario B. A feasible min-r articulated probe trajectory exists such that ab of the trajectory passes through an obstacle endpoint u, and bt of the trajectory passes through an obstacle endpoint v, where $u, v \in V$ and $u \neq v$. Recall that $\angle cbt$ of the trajectory is less than or equal to $\pi/2$ radians. Without loss of generality, assume that segment bc of the probe is rotated clockwise around b to reach t, as in Figure 7 (the other case is symmetrical).

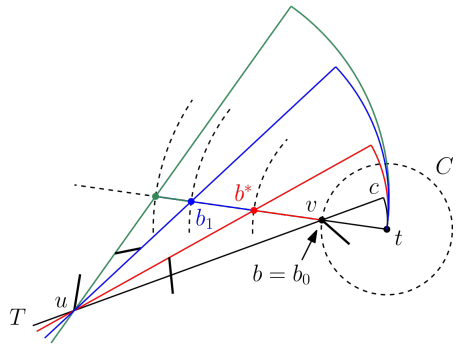


Figure 7: Finding the extremal feasible min-r articulated probe trajectory in Scenario B.

In this case, the minimum value of r for a feasible trajectory occurs when $b = v$. Let b_0 denote that location of b, and T be the corresponding trajectory. If T is free of obstacles, then T is a feasible min-r articulated trajectory (case IV of the lemma).

We now assume otherwise. Let ρ_{b_0} denote the *reversal* (i.e., opposite in direction) of the ray emanating from b_0 passing through t. We increase r by moving b away from b_0 along ρ_{b_0}, while maintaining the intersection of ab with u and that of bt with v, until the trajectory becomes feasible. Observe the following: i) If σ_{bct} of T intersects any obstacle line segment, then for certain there is no feasible articulated trajectory that intersects u outside C and v inside C. So, σ_{bct} of T must be empty of obstacle line segments. ii) If bt, bc, or γ_{ct} intersects an obstacle line segment at any given moment during the process of increasing r, then the trajectory would never become feasible thereafter.

These observations imply that, when $b = b_0$, ab of T must be intersected by some obstacle line segment. By increasing r, we may rid the trajectory of obstacle line segments that intersect ab. Let b^* denote the closest point to b_0 on ρ_{b_0} for which the corresponding articulated trajectory is feasible. Note that ab, at the moment, intersects an obstacle endpoint (case V of the lemma), as illustrated by the red trajectory in Figure 7.

Remark. Observe that we can continue to increase r, while still having a feasible articulated trajectory, until b reaches some point b_1, at which either i) ab, bc, or γ_{ct} collides with an obstacle line segment, or ii) $\angle cbt = \pi/2$. Let r_{b_0}, r_{b^*}, and r_{b_1} be the lengths of bc when $b = b_0$, $b = b^*$, and $b = b_1$, respectively, where $r_{b_0} \leq r_{b^*} \leq r_{b_1}$. In addition, let $r_{\pi/2}$ be the length of bc when $\angle cbt = \pi/2$. According to our earlier arguments, $[r_{b^*}, r_{b_1}]$ is a feasible contiguous subset of $[r_{b_0}, r_{\pi/2}]$. In fact, there could exist multiple (disjoint) contiguous feasible subsets of $[r_{b_0}, r_{\pi/2}]$, given that ab may enter and leave intersections with multiple obstacle line segments during the process of increasing r, while σ_{bct} remains free of obstacle line segments (refer to the blue and green trajectories in Figure 7 for an instance).

This concludes the proof of Lemma 1.

B Characterizing infeasible final configuration space

Let C' be the circle centered at t and of radius $\sqrt{2}r$. A circular sector associated with a final configuration can only intersect an obstacle line segment lying inside C'. Instead of characterizing the lower and upper bounds of θ_C as θ_S varies from 0 to 2π (as in Section 3.2), here we perform the characterization the other way around. For conciseness, we only present arguments for the negative half of the θ_S-range, which is $[\theta_C - \cos^{-1} r/R, \theta_C]$; similar arguments apply to the other half due to symmetry. We have two cases, depending on whether an obstacle line segment s lies 1) inside C or 2) outside C and inside C'.

Case 1. Obstacle line segment s inside C. For brevity, the quarter circular sector associated with a point b (i.e., the maximum possible area swept by segment bc to reach t), where the angle of tb (relative to the x-axis) is θ_C, is referred to as the *quart-sector of* θ_C.

We define ϕ_1, ϕ_2, and ϕ_3 as follows (Figure 8A). ϕ_1 is the smallest angle θ_C at which the circular arc of the quart-sector of θ_C intersects s (at one of its endpoints or interior points). ϕ_2 is the smallest angle θ_C at which bt of the quart-sector of θ_C intersects s (at one of its endpoints). ϕ_3 is the largest angle θ_C at which bt of the quart-sector of θ_C intersects s (at one of its endpoints). Observe that, as θ_C varies from 0 to 2π, ϕ_1 and ϕ_3 are the angles θ_C at which the quart-sector of θ_C first and last intersects s, respectively.

We are only concerned with finding the lower bound of θ_S for $\theta_C \in [\phi_1, \phi_2]$, since the entire negative half of the θ_S-range (i.e., $[\theta_C - \cos^{-1} r/R, \theta_C]$) is feasible for $\theta_C \in [0, \phi_1] \cup [\phi_3, 2\pi)$, and is infeasible for $\theta_C \in [\phi_2, \phi_3]$ due to intersection of bt with s (Figure 8A).

For $\theta_C \in [\phi_1, \phi_2]$, the lower bound of θ_S can be represented by a piecewise continuous curve, which consists of at most two pieces, corresponding to two intervals $[\phi_1, \alpha]$ and $[\alpha, \phi_2]$, where α is the angle θ_C of the intersection point between C and the supporting line of s. If $\phi_1 \leq \alpha$, then the curve has two pieces; otherwise, the curve is of one single piece.

For $\theta_C \in [\phi_1, \alpha]$, the lower bound of θ_S is indicated by the endpoint a of line segment abc', where c' is the intersection point between s and the circular arc centered at b (Figure 8B). If no intersection occurs between s and the circular arc,

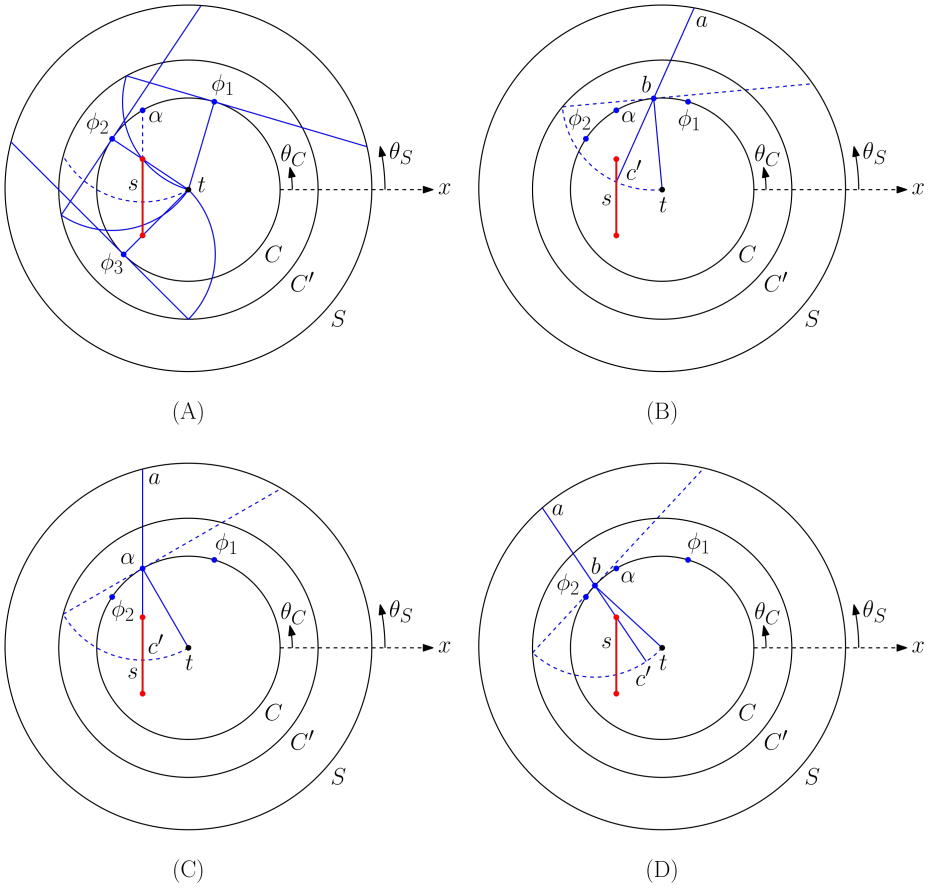


Figure 8: Infeasible final configurations due to an obstacle line segment s inside C. Illustrations of θ_S-lower bounds for (A) $\theta_C \in [\phi_1, \phi_2]$, (B) $\phi_1 < \theta_C < \alpha$, (C) $\theta_C = \alpha$, and (D) $\alpha < \theta_C < \phi_2$.

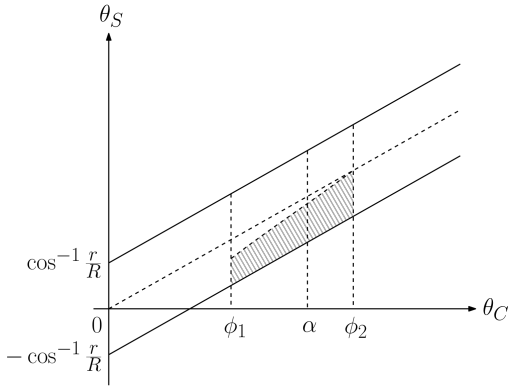


Figure 9: Infeasible final configuration space due to an obstacle line segment s inside C.

then the lower bound of θ_S is given by the endpoint a of line segment abc', where bc' intersects an endpoint of s.

For $\theta_C \in [\alpha, \phi_2]$, the lower bound of θ_S is indicated by the endpoint a of line segment abc', where bc' intersects an endpoint of s (Figure 8D). The lower bound of θ_S is equal to θ_C when $\theta_C = \phi_2$. See Figure 9 for a sketch of the infeasible final configuration space.

Case 2. Obstacle line segment s outside C and inside C'. As depicted in Figure 10, we only need to worry about computing the lower bound of θ_S for $\theta_C \in [\phi_1, \phi_2]$, given that the entire negative half of the θ_S-range (i.e., $[\theta_C - \cos^{-1} r/R, \theta_C]$) is feasible for $\theta_C \in [0, \phi_1] \cup [\phi_2, 2\pi)$. The analysis is similar to case 1 and thus omitted herein. A sketch of the corresponding infeasible final configuration space is shown in Figure 11.

Observe that any of the curves just described for characterizing the lower or upper bound of θ_S can be computed in constant time. Thus, given an obstacle line segment s, the associated infeasible final configuration space can be found in $O(1)$ time. As a result, it takes $O(n)$ time to determine the infeasible final configuration space for n obstacle line segments.

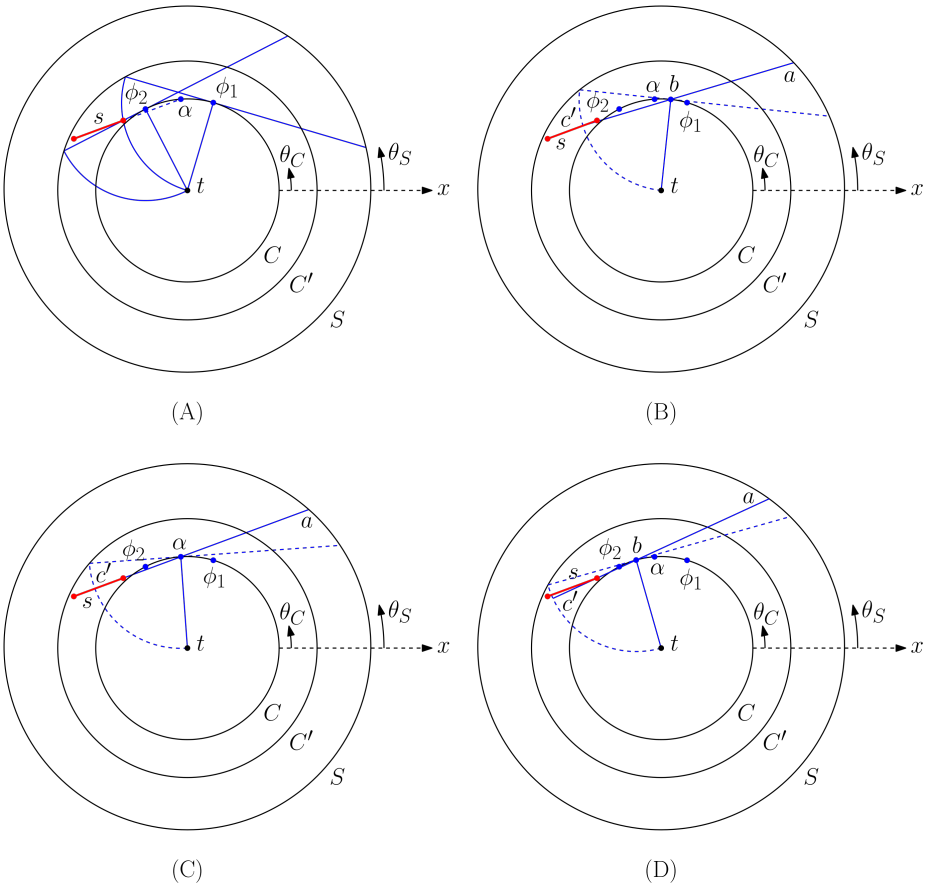


Figure 10: Infeasible final configurations due to an obstacle line segment s outside C and inside C'. Illustrations of θ_S-lower bounds for (A) $\theta_C \in [\phi_1, \phi_2]$, (B) $\phi_1 < \theta_C < \alpha$, (C) $\theta_C = \alpha$, and (D) $\alpha < \theta_C < \phi_2$.

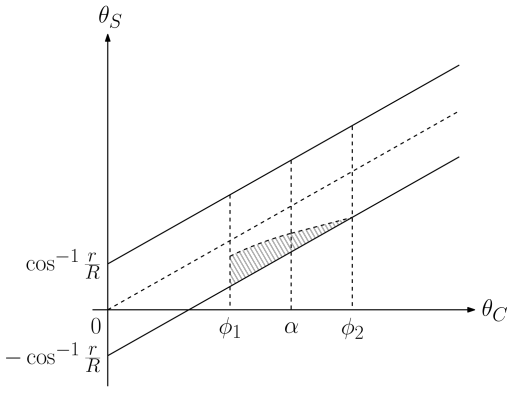


Figure 11: Infeasible final configuration space due to a line segment s outside C and inside C'.

Dynamic Products of Ranks

David Eppstein*

Abstract

We describe a data structure that can maintain a dynamic set of points given by their Cartesian coordinates, and maintain the point whose product of ranks within the two coordinate orderings is minimum or maximum, in time $O(\sqrt{n \log n})$ per update.

1 Introduction

The *rank* of an element in a collection of elements is its position in a list of all elements, sorted by some associated numerical value. If elements have a multidimensional vector of values associated with them, then each of these values gives rise to a different rank, and we may wish to aggregate these multiple ranks into a single combined score. One common method of aggregating ranks is to use the geometric mean or equivalently the product of ranks as the combined score. This method is used in applications ranging from finding differentially regulated genes in DNA microarray data [2], choosing winners in multi-discipline sports events [7], and measuring the scholarly output of economists [10] to image recognition [8] and spam filtering in web search engines [6].

In many of these applications, it is natural for the elements in the collection and their associated numerical values to change dynamically, and when they do the whole system of ranks for other elements may change. For instance, inserting one new element, with a low numerical value, will increase the ranks of all elements with larger values. This raises the question: how can we update the elements and their numerical values, and maintain information about the product of ranks?

We can model this as a geometry problem, in which the elements in the collection are modeled as points in the Cartesian plane, with the x- and y-coordinates of these points representing their associated numerical values. In this model, we would like to maintain a dynamic set of pairs of real numbers, subject to point insertion and point deletion, and as we do so, maintain dynamically the point whose product of ranks in the two coordinate orderings is minimum or maximum.

In this work we provide a solution to this dynamic product of ranks problem. We solve the dynamic product of ranks problem, in the special case when there are two rankings being combined, in time $O(\sqrt{n \log n})$ per update.

There are three main ideas to our method:

- We partition the points into *rigid* subsets: sets of points whose ranks all change in lockstep with each operation (that is, without changing the difference between the ranks of any two elements in the set). Our partition will have the property that each update will rebuild rigid subsets of total size $O(\sqrt{n \log n})$ and search for the point with minimum or maximum product of ranks within $O(\sqrt{n/\log n})$ of these subsets.
- We provide two solutions to the dynamic product of ranks problem within each rigid subset. One solution applies a lifting transformation (to the pairs of ranks of the points, not their given coordinates) to turn it into a problem of querying a (static) three-dimensional convex hull. Dually, the other solution uses analogues of the classical Voronoi diagram and farthest-point Voronoi diagram, minimization and maximization diagrams with convex-polygon cells.
- We provide linear-time constructions for the lifted convex hull in the minimization version of the problem, and for the maximization diagram in the maximization version of the problem, adapted from two different algorithms for linear-time construction of Voronoi diagrams of points in convex position.

Our method can be generalized to larger numbers of rankings, but with a quadratic blowup in the dimension of the lifting transformation that (together with the high complexity of higher-dimensional extreme point queries) leads to a running time per update that is only slightly smaller than the trivial naive algorithm of updating all rankings and recomputing all products in linear time per update. For this reason, we restrict our attention to maintaining information about the product of two rankings.

2 Rigid subsets

2.1 Lifted hull

We say that a subset S of elements in our product of ranks problem is *rigid*, through a sequence of updates, if none of the updates performs an insertion or deletion

*Computer Science Department, University of California, Irvine, eppstein@uci.edu. This work was supported in part by the US National Science Foundation under grant CCF-1616248.

of an element of S, or of another element whose position in either of the two rankings lies between two elements of S. Equivalently, the difference in ranks of any two elements of S remains invariant throughout the given sequence of updates.

Lemma 1 *Let S be any subset of elements in the product of ranks problem, of size m. Then in time $O(m \log m)$ we can build a data structure for S such that, throughout any sequence of updates for which S is rigid, we can compute the elements of S with the minimum or maximum product of ranks in time $O(\log m)$ per update.*

Proof. Let (x_i, y_i) be the ranks of the elements of S prior to the sequence of updates for which S is rigid. We construct the three-dimensional convex hull of the lifted points $(x_i, y_i, x_i y_i)$, and a Dobkin–Kirkpatrick hierarchy allowing us to perform linear optimization queries (finding the extreme point on the resulting hull of a given linear function) in time $O(\log m)$ per query [5]. The hull takes $O(n \log n)$ time to construct and its Dobkin–Kirkpatrick hierarchy takes an additional $O(n)$ time. For each element, let $z_i = x_i y_i$ denote its third coordinate in this lifted point set.

After a sequence of updates that have changed the ranks by subtracting the same offset a from each rank x_i and the same offset b from each rank y_i within S, the updated products of ranks are

$$(x_i - a)(y_i - b) = ab - a y_i - b x_i + z_i,$$

a linear function of the three coordinates of the lifted points, so the elements with the minimum and maximum product of ranks can be found by a linear optimization query. □

This method is closely analogous to the classical lifting transformation of two-dimensional closest-point problems to three-dimensional extreme-point problems [3], which in its most commonly used form maps pairs (x_i, y_i) to triples $(x_i, y_i, x_i^2 + y_i^2)$; however, we use a different quadratic function for the third coordinate. Note that we will only query this structure for pairs (a, b) with $a \le x_i$ and $b \le y_i$, because the differences $x_i - a$ and $y_i - b$ represent ranks and are therefore non-negative.

2.2 Linear time construction

To construct the lifted hull more quickly, it is helpful to reduce the set of points to a subset whose projection to the plane is convex.

Lemma 2 *Let S be a set of points, let (a, b) be a pair of numbers with a less than or equal to all x-coordinates in S and b less than or equal to all y-coordinates in S. Let (x_i, y_i) be the point in S minimizing $(x_i - a)(y_i - b)$. Then (x_i, y_i) lies on the convex hull of S.*

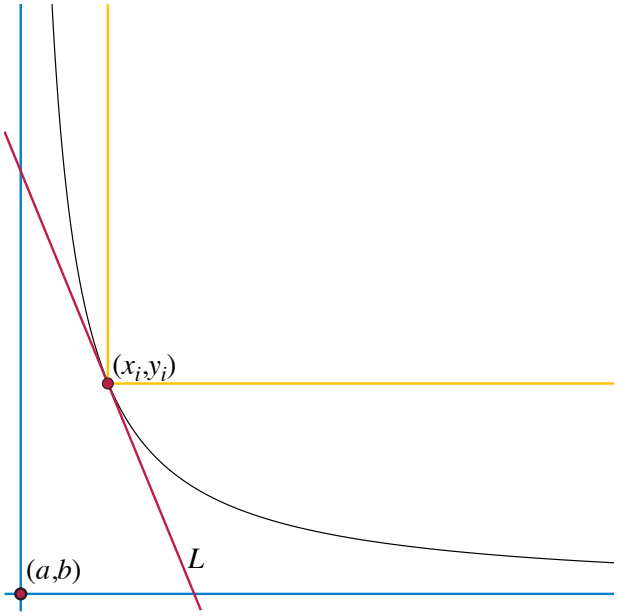


Figure 1: The minimizer of $(x_i - a)(y_i - b)$ must be a convex hull vertex, because the region below line L, the tangent to the hyperbola through (x_i, y_i), must be disjoint from S (Lemma 2). Analogously, the maximizer of $(x_i - a)(y_i - b)$ must be a maximal point of S, because the region above and to its left (yellow) must be disjoint from S (Lemma 5).

Proof. The locus of points (x, y) with $(x - a)(y - b) = (x_i - a)(y_i - b)$ is a hyperbola, asymptotic to the lines $x = a$ and $y = b$, with (x_i, y_i) on its positive branch. Let L be the line tangent to this hyperbola at (x_i, y_i); see Figure 1. Then the halfplane below L must be disjoint from S, for any point (x_j, y_j) between L and the other branch of the hyperbola would have a smaller value of $(x_j - a)(y_j - b)$ and by the assumptions on a and b there are no points of S on the other side of the other branch of the hyperbola. □

Aggarwal et al. [1] showed that, for 3d points whose two-dimensional projection is convex, the 3d convex hull can be constructed in linear time. In the next lemma we apply this result to the lifted hull of Lemma 1.

Lemma 3 *Let S be any subset of elements in the product of ranks problem, of size m, for which the sorted order by x-coordinate is known. Then in time $O(m)$ we can build a data structure for S such that, throughout any sequence of updates for which S is rigid, we can compute the elements of S with the minimum product of ranks in time $O(\log m)$ per update.*

Proof. We use Graham scan to compute the 2d convex hull from the sorted order of points in linear time, and the algorithm of Aggarwal et al. [1] to compute the 3d convex hull from the 2d convex hull in linear time. The

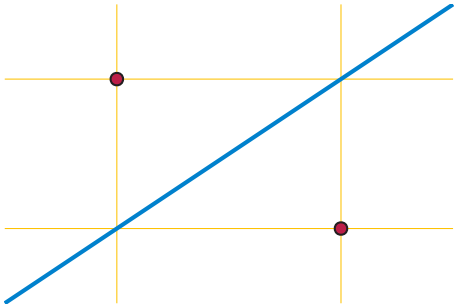


Figure 2: The bisector between two sites in the minimization diagram is the line through the other two corners of their bounding box.

Dobkin–Kirkpatrick hierarchy construction time is also linear. □

2.3 Maximization diagram

Instead of lifting the points (x_i, y_i) to the convex hull of three-dimensional points $(x_i, y_i, x_i y_i)$, an alternative representation for each rigid subset would be to represent it by the minimization diagram or maximization diagram of the functions $f_i(a, b) = (x_i - a)(y_i - b) = ab - ay_i - bx_i + z_i$. Then, the minimum or maximum product of ranks for the rigid subset with rank offsets a and b could be obtained by performing a point location query in this diagram, rather than by performing an extreme-point query on a three-dimensional polyhedron.

Because the quadratic term ab in the definition of the function $f_i(a, b)$ does not depend on the point (x_i, y_i), and is equal for all points, it does not affect the minimization or maximization: we obtain the same minimization or maximization diagrams for the linear functions $g_i(a, b) = -ay_i - bx_i + z_i$. As the minimization or maximization diagram of linear functions, these diagrams have convex polygon cells, separated by *bisector* lines, the lines consisting of the points (a, b) at which two of these functions are equal.

Lemma 4 *The bisector of any two given points (sites) (x_i, y_i) and (x_j, y_j) in the minimization or maximization diagram described above is a line that passes through the other two corners (x_i, y_j) and (x_j, y_i) of the bounding box of the two points (Figure 2).*

Proof. When the bounding box is a square, this follows by symmetry: a reflection through the line described in the lemma maps the two given points to each other, swapping the two Cartesian coordinates, so for any point (a, b) on the line described in the lemma, the coordinate differences between (a, b) and the two given points are equal but reversed. That is, $|x_i - a| = |y_j - b|$ and $|x_j - a| = |y_i - b|$. Since the quantity being minimized is the product of these coordinate differences, it is equal for the two given points along this line.

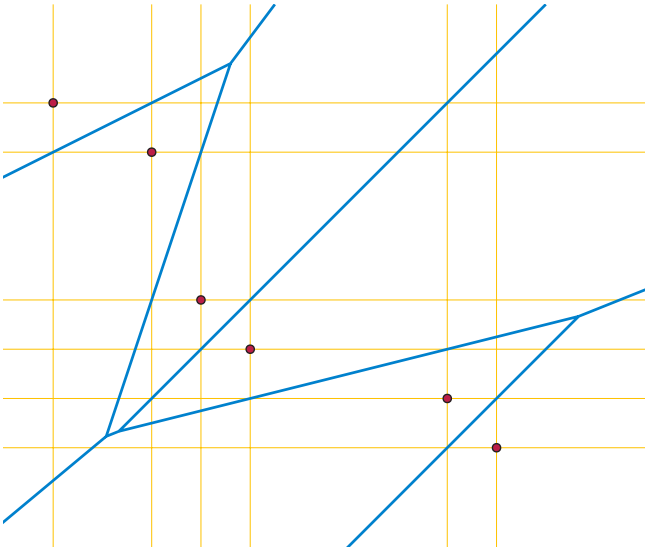


Figure 3: The maximization diagram for a given set of maximal points. Although this diagram is well-defined over the whole plane, we will only query it within the bottom-left quadrant, below and to the left of all the given points.

For any other two points, not both on the same vertical or horizontal line, we may apply a linear transformation to one of the coordinates that makes the bounding box a square; this transformation affects both of the functions g_i and g_j in the same way, so the bisector of the transformed points (the diagonal of the square) is the transformation of the bisector, which must therefore be the diagonal of the original bounding box. The remaining case, that the points are on a horizontal or vertical line, follows by continuity. □

Figure 3 depicts an example of the maximization diagram described above.

2.4 Expected linear time construction

These diagrams can be constructed in $O(n \log n)$ time, either by interpreting them as a lower or upper envelope of three-dimensional planes (the graphs of the functions they minimize or maximize) or by using algorithms for abstract Voronoi diagrams with bisectors determined as in Lemma 4 [9]. However, as we now show, they can be constructed in expected linear time.

Our construction begins with the following analogue of Lemma 2. We observe that, in constructing the maximization diagram for a collection of points, we need only include the points (x_i, y_i) that are maximal (meaning that there is no other point (x_j, y_j) with $x_j \geq x_i$ and $y_j \geq y_i$), for those are the only ones that can produce the maximum of the function values at any point (a, b).

Lemma 5 *Let S be a set of points, let (a, b) be a pair of numbers with a less than or equal to all x-coordinates in S and b less than or equal to all y-coordinates in S. Let (x_i, y_i) be the point in S maximizing $(x_i - a)(y_i - b)$. Then (x_i, y_i) is one of the maximal points of S, meaning that there is no other point (x_j, y_j) in S with $x_j \geq x_i$ and $y_j \geq y_i$.*

Proof. Any such point (x_j, y_j) would have a larger value of $(x_i - a)(y_i - b)$. □

The quarter-plane of points with larger x- and y-coordinates than (x_i, y_i), and their relation to the hyperbola of points with equal query values to (x_i, y_i), is shown in Figure 1.

To construct the maximization diagram in expected linear time we adapt an algorithm by Paul Chew for Voronoi diagrams of convex polygons [4].

Lemma 6 *Let S be a set of points, all of which are maximal in S, indexed in sorted order by their x-coordinates, and let (x_i, y_i) and (x_{i+1}, y_{i+1}) be consecutive points in this ordering. Then in the maximization diagram for $(x_i - a)(y_i - b)$, the cells for these two points share an edge.*

Proof. Within the bounding rectangle of (x_i, y_i) and (x_{i+1}, y_{i+1}), the point (x_i, y_i) has a larger query value than all points of S with smaller index, and the point (x_{i+1}, y_{i+1}) has a larger query value than all points of S with larger index, so the maximization diagram within the rectangle consists only of points in the cells for these two points. By Lemma 4 the cells meet within the rectangle along the bisector of these two points, which is the diagonal of the rectangle. □

The shared edge is not in a part of the diagram that we will query in our data structure for products of ranks, but its location is unimportant for the use we will make of it in the following lemma.

Lemma 7 *Let S be any subset of elements in the product of ranks problem, of size m, for which the sorted order by x-coordinate is known. Then in randomized expected time $O(m)$ we can build a data structure for S such that, throughout any sequence of updates for which S is rigid, we can compute the elements of S with the maximum product of ranks in time $O(\log m)$ per update.*

Proof. The maximal points in S can be found in linear time from the sorted order by x-coordinates, using a stack algorithm closely related to Graham scan.

We construct the maximization diagram by a randomized incremental algorithm in which we randomly permute the points and add them to the diagram one at a time in that random order. By the analysis of Chew [4], this can be done in expected constant time per point as long as we know the identity of a neighboring cell in the diagram of the points added so far. We can form a random permutation with this additional information about neighboring cells by starting with a doubly linked list of all of the points, in x-coordinate order, deleting randomly chosen points from the linked list until none are left, and then reversing the order of the deletions. By Lemma 6, the neighbors of a point (x_i, y_i) in the linked list at the time of its deletion will form neighboring cells in the maximization diagram at the time of its insertion.

Because it is the maximization diagram of a set of linear functions, we can interpret this diagram as a three-dimensional intersection of halfspaces, and construct a Dobkin–Kirkpatrick hierarchy from it in linear time, suitable for performing point location queries in logarithmic time. (Alternatively, the history DAG of a vertical decomposition of the randomized incremental maximization diagram construction can be used as a point location data structure with logarithmic expected time per query.) □

3 Partitioned data structure

3.1 One-dimensional partition

To partition our given elements into rigid subsets, we first consider a one-dimensional partition method, which we will apply separately to the two rankings of the elements.

Lemma 8 *Let f be any positive concave function of a single argument. Then for any sequence S of ordered values undergoing insertions and deletions, we can maintain a partition of S into an ordered sequence of $O(n/f(n))$ contiguous subsets, with $O(f(n))$ elements in each subset, changing $O(1)$ subsets per update, using a data structure with time $O(\log n)$ per update, where n denotes the current size of S.*

Proof. We use binary search trees to keep track of the sequence of elements and the sequence of subsets. As keys for the binary search tree of subsets, we use the values of their first elements. In this way we can find the subset containing the updated element, after any update, and determine the new size of this subset. We also keep track of the sizes of each subset and maintain priority queues for the largest subsets and for the smallest consecutive pairs of subsets.

We maintain as an invariant the requirements that the sizes of all subsets in the partition are at most $2\lceil f(n)\rceil + 2$, and that no two consecutive subsets both have size less than $\lceil f(n)\rceil - 1$. We say that our structure is *growing* if, for the most recent update having a different value of $\lceil f(n)\rceil$, that value was smaller than the current value, and *shrinking* otherwise. If the structure

is growing, we require that all subsets have size at most $2\lceil f(n)\rceil$, and if it is shrinking, we require that no two consecutive subsets both have size less than $\lceil f(n)\rceil$.

On each update, if the structure is growing, we select an arbitrary pair of consecutive subsets of size $\lceil f(n)\rceil - 1$ (if such a pair exists) and merge them into a single subset. If the structure is shrinking, we select an arbitrary subset of size greater than $2\lceil f(n)\rceil$ (if such a subset exists) and split it into two subsets of size as close to equal as possible. We claim that this is sufficient to maintain our invariants. Clearly, it does so at the updates for which $\lceil f(n)\rceil$ does not change, so we need only consider the steps at which it does change.

In the case that $\lceil f(n)\rceil$ changes in such a way that the structure was growing before the update and is shrinking after the update, the invariants are automatically maintained, because the ranges of sizes of subsets and consecutive pairs of subsets that are allowed remain unchanged. The same is true when the structure was shrinking before the update and growing after the update.

When $\lceil f(n)\rceil$ increases twice in a row (so that it was growing both before and after the second increase), let n_0 be the value of n at the first increase. Then at that time, there must be at most $n_0/f(n_0)$ consecutive pairs of small subsets, and (by concavity of f) at least $n_0/f(n_0)$ steps between the two increases. It only takes $n_0/2f(n_0)$ steps to eliminate all of the consecutive pairs of small subsets. So by the time that the second increase happens, all of the consecutive pairs of small subsets will have been eliminated, maintaining the invariant.

Similarly, when $\lceil f(n)\rceil$ decreases twice in a row (so that it was shrinking both before and after the second decrease), let n_0 be the value of n at the first decrease. Then at that time, there must be at most $n_0/2f(n_0)$ large subsets, and (by concavity of f) at least $n_0/f(n_0)$ steps between the two decreases. It only takes $n_0/2f(n_0)$ steps to eliminate all of the large subsets. So by the time that the second decrease happens, all of the consecutive pairs of large subsets will have been eliminated, maintaining the invariant. □

3.2 Two-dimensional partition

We now use our one-dimensional rank partition to partition the given elements into subsets, most of which remain rigid in each update. If the ranks of each element are (x_i, y_i), we will maintain one rank partition on the ranks x_i, and a second rank partition on the ranks y_i, each with parameter $f(n) = \sqrt{n \log n}$. Then each subset S_k of our two-dimensional partition will consist of elements that are grouped together both in the partition on the x-ranks and in the partition on the y-ranks.

Lemma 9 *The partition into subsets S_k described above has the following properties:*

- *There are $O(n/\log n)$ subsets.*
- *Each update to the data causes $O(\sqrt{n/\log n})$ of the subsets, with total size $O(\sqrt{n \log n})$, to be non-rigid.*
- *Each update to the data causes $O(\sqrt{n/\log n})$ of the subsets, with total size $O(\sqrt{n \log n})$, to be replaced by new subsets due to the change in the underlying one-dimensional partitions.*

Proof. It follows from Lemma 8 and our choice of the function f that each one-dimensional partition has $O(\sqrt{n/\log n})$ subsets, of size $O(\sqrt{n \log n})$, and that each update causes $O(1)$ changes to the one-dimensional partition. Because each subset in the two-dimensional partition is determined by a pair of subsets in the two one-dimensional partitions, there are $O(n/\log n)$ subsets in the two-dimensional partition.

In any update, only one subset of each one-dimensional partition contains non-rigid subsets of the two-dimensional partition. Therefore, the total number of non-rigid subsets is at most twice the number of two-dimensional subsets that can be contained in a single one-dimensional subset, $O(\sqrt{n/\log n})$, and the total size of the non-rigid subsets is at most twice the size of a one-dimensional subset, $O(\sqrt{n \log n})$. The analysis of the number of subsets that are replaced with new subsets and their total size is similar: each change to a one-dimensional subset causes changes to $O(\sqrt{n/\log n})$ two-dimensional subsets having a total of $O(\sqrt{n \log n})$ elements, so the bounds on replaced subsets follow from the fact that each update causes $O(1)$ changes to the one-dimensional partitions. □

4 Which subsets to query?

We introduced Lemma 2 and Lemma 5 to aid in the efficient construction of rigid subsets, but they can also be used to reduce the number of rigid subsets that we must query after any update. As these two lemmas show, the point with the smallest product of ranks must be minimal in the coordinate ordering of the points, and the point with the largest product of ranks must be maximal. The two-dimensional partition of Lemma 9 partitions the points in a grid pattern, and we need only query the rigid subsets for cells in this grid that can contain minimal or maximal points.

Lemma 10 *Let a given set of points be partitioned by k axis-parallel lines into a grid of cells, represented in such a way that in constant time we can find the neighboring cell in any direction from any given cell and find the lowest nonempty cell in any column of the grid. Then in time $O(k)$ we can identify a subset of $O(k)$ of the grid cells that contain all of the minimal points in the set.*

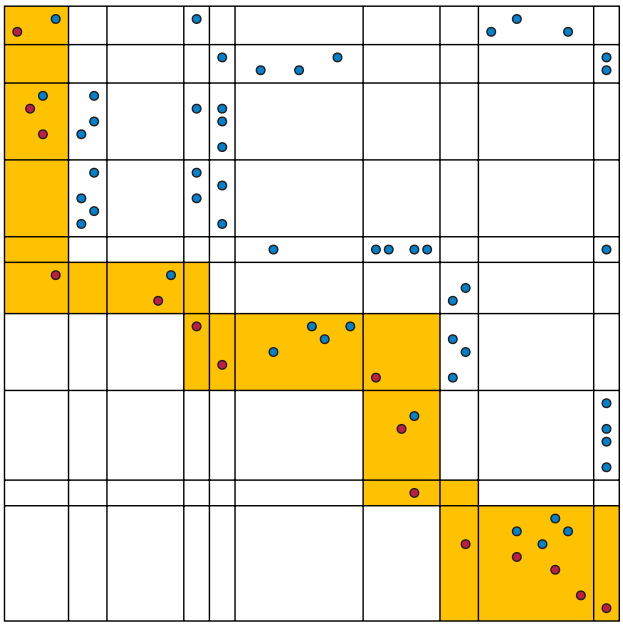


Figure 4: A grid partition of a point set, and a path (yellow shading) through the cells of the grid, such that the cells of the path contain all minimal points of the set (shown as red).

Proof. As we describe below, we select cells in the grid along a path from top left to bottom right, such that every unselected cell below the path is also below the lowest nonempty cell in its column, and every unselected cell above the path has a nonempty selected cell below and to the left of it. In this way, every minimal point of the given point set belongs to a selected cell, for there can be no points below and to the left of the path, and all points above and to the right are not minimal. Figure 4 shows an example.

To find this path of grid cells, we begin at the top left cell of the grid. Then we repeatedly step to a neighboring cell, according to the following rules:

- If the current cell is the bottom right cell of the grid, we terminate the path.
- If the current cell is not the lowest nonempty cell in its column, or if it belongs to the rightmost column, we step to the next cell down.
- Otherwise, we step to the next cell to the right.

The path must extend across all columns, for it can only stop in the rightmost column. If a cell is below the path, it must also be below the lowest nonempty cell in its column, or we would have stepped downward to it when the path crossed its column; therefore, all cells below the path are empty. If a cell is above the path, then the path must have stepped below it in some column to the left of it, which can only happen when the lowest nonempty cell in that column is below and to the left of the given cell. Therefore, all cells above the path have a nonempty cell below and to the left of them. □

A similar method, with the ability to find the highest nonempty cell in each column, can find a path of grid cells containing all maximal points.

5 Overall data structure

Our overall data structure consists of:

- Two binary search trees on the two coordinate values of the elements, augmented to allow the rank of any element at any step of the update sequence to be looked up in logarithmic time per query.
- Two one-dimensional partitions of the elements, one for each of the two rankings of the elements, according to Lemma 8, with the parameter choice specified for Lemma 9.
- The two-dimensional partition of the elements into rigid subsets S_k defined from these one-dimensional partitions, according to Lemma 9.
- A graph describing the relation between neighboring cells in this two-dimensional partition, and the lowest or highest nonempty cell in each column of cells, suitable for use in Lemma 10.
- A sorted list of points in each partition set, sorted by their x-coordinates.
- A data structure for maintaining the extreme points for the product of ranks of each subset S_k, through updates for which it is rigid, according to Lemma 1.

Theorem 11 *The data structure described above can maintain the minimum or maximum product of ranks in time $O(\sqrt{n \log n})$ per update for the minimimum, or the same time bound in expectation for the maximum.*

Proof. By Lemma 9, each update causes changes to subsets S_k of total size $O(\sqrt{n \log n})$; by Lemma 3 and Lemma 7, reconstructing the extreme-point data structures for these subsets takes the stated time per update. After each update, we may use Lemma 10 to find a subset of $O(\sqrt{n/\log n})$ subsets to query, use the binary search trees to determine the offsets in rank for each of these selected subsets, and then query the extreme point within each subset in time $O(\log n)$ by Lemma 1. The total time for these queries is again the stated time per update. Maintaining the binary search trees and one-dimensional partitions takes an amount of time that is negligible with respect to this total time bound. □

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 & Computational Geometry*, 4(6):591–604, 1989. doi:10.1007/BF02187749.

[2] Rainer Breitling, Patrick Armengaud, Anna Amtmann, and Pawel Herzyk. Rank products: a simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments. *FEBS Letters*, 573(1-3):83–92, 2004. doi:10.1016/j.febslet.2004.07.055.

[3] Kevin Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228, December 1979. doi:10.1016/0020-0190(79)90074-7.

[4] L. Paul Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dartmouth College Department of Mathematics and Computer Science, 1990. URL: https://www.cs.dartmouth.edu/~trdata/reports/TR90-147.pdf.

[5] David P. Dobkin and David G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, 6(3):381–392, 1985. doi:10.1016/0196-6774(85)90007-0.

[6] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*. ACM, 2001. doi:10.1145/371920.372165.

[7] International Federation of Sport Climbing. Rules 2019, March 2019. URL: https://www.ifsc-climbing.org/images/World_Competitions/IFSC-Rules_2019_v192_PUBLIC.pdf.

[8] Chao Li and A. Barreto. An integrated 3D face-expression recognition approach. In *Proceedings of the International Conference on Acoustics Speed and Signal Processing*. IEEE, 2006. doi:10.1109/icassp.2006.1660858.

[9] K. Mehlhorn, St. Meiser, and C. Ó'Dúnlaing. On the construction of abstract Voronoi diagrams. *Discrete & Computational Geometry*, 6(3):211–224, 1991. doi:10.1007/BF02574686.

[10] Christian Zimmermann. Academic rankings with RePEc. *Econometrics*, 1(3):249–280, December 2013. doi:10.3390/econometrics1030249.

Closest-Pair Queries and Minimum-Weight Queries are Equivalent for Squares

Abrar Kazi* Michiel Smid†

Abstract

Let S be a set of n weighted points in the plane and let R be a query range in the plane. In the range closest pair problem, we want to report the closest pair in the set $R \cap S$. In the range minimum weight problem, we want to report the minimum weight of any point in the set $R \cap S$. We show that these two query problems are equivalent for query ranges that are squares, for data structures having $\Omega(\log n)$ query times. As a result, we obtain new data structures for range closest pair queries with squares.

1 Introduction

Let S be a set of n points in the plane. In the *range closest pair problem*, we want to store S in a data structure, such that for any axes-parallel query rectangle R, the closest pair in the point set $R \cap S$ can be reported. This problem has received considerable attention; see [1, 2, 3, 6, 7, 9, 10, 11, 12]. The best known result is by Xue *et al.* [12], who obtained a query time of $O(\log^2 n)$ using a data structure of size $O(n \log^2 n)$. For the special case when the query range R is a *square* (or, more generally, a fat rectangle), Bae and Smid [2] showed that a query time of $O(\log n)$ is possible, using $O(n \log n)$ space.

Assume that each point p of S has a real weight $\omega(p)$. In the *range minimum weight problem*, we want to store S in a data structure, such that for any axes-parallel query rectangle R, the minimum weight of any point in $R \cap S$ can be reported. Using a standard range tree of size $O(n \log n)$, such queries can be answered in $O(\log^2 n)$ time; see, e.g., de Berg *et al.* [5]. Chazelle [4] showed the following results for such queries on a RAM: (i) for every $\varepsilon > 0$, $O(\log^{1+\varepsilon} n)$ query time using $O(n)$ space, (ii) $O(\log n \log\log n)$ query time using $O(n \log\log n)$ space, and (iii) for every $\varepsilon > 0$, $O(\log n)$ query time using $O(n \log^{\varepsilon} n)$ space. We are not aware of better solutions for query squares.

*School of Computer Science, Carleton University, Ottawa, Canada, AbrarKazi@cmail.carleton.ca. Research supported by an NSERC Undergraduate Student Research Award.

†School of Computer Science, Carleton University, Ottawa, Canada, michiel@scs.carleton.ca. Research supported by NSERC.

1.1 Our Results

We show that the range closest pair problem and the range minimum weight problem are equivalent for query *squares*[1], for data structures having $\Omega(\log n)$ query times. We say that a function f is *smooth*, if $f(O(n)) = O(f(n))$. Our main results are as follows:

Theorem 1 *Let M and Q be smooth functions such that $M(n) \geq n$ and $Q(n) = \Omega(\log n)$. Assume there exists a data structure of size $M(n)$ that answers a range minimum weight query, for any query square, in $Q(n)$ time. Then there exists a data structure of size $O(M(n))$ that answers a range closest pair query, for any query square, in $O(Q(n))$ time.*

Theorem 2 *Let M and Q be smooth functions such that $M(n) \geq n$ and $Q(n) = \Omega(\log n)$. Assume there exists a data structure of size $M(n)$ that answers a range closest pair query, for any query square, in $Q(n)$ time. Then there exists a data structure of size $O(M(n))$ that answers a range minimum weight query, for any query square, in $O(Q(n))$ time.*

Theorem 1, together with the above mentioned results of Chazelle, imply the following:

Corollary 3 *Let S be a set of n points in the plane. Range closest pair queries, for any query square, can be answered*

1. *in $O(\log^{1+\varepsilon} n)$ time using $O(n)$ space,*
2. *in $O(\log n \log\log n)$ time using $O(n \log\log n)$ space,*
3. *in $O(\log n)$ time using $O(n \log^{\varepsilon} n)$ space.*

Observe that the third result in Corollary 3 improves the space bound in Bae and Smid [2] from $O(n \log n)$ to $O(n \log^{\varepsilon} n)$.

Our proofs of Theorems 1 and 2 are based on the approach of Bae and Smid [2] for range closest pair queries with squares. Their solution uses data structures for (i) deciding whether a query square contains at most c points of S, for some fixed constant c, (ii) computing the smallest square that has a query point as its bottom-left corner and contains c' points of S, for some fixed constant c', and (iii) range minimum weight queries with

[1] throughout this paper, squares are always axes-parallel

squares. They showed that the queries in (i) and (ii) can be answered in $O(\log n)$ time using $O(n \log n)$ space. We will improve the space bound for both these queries to $O(n)$.

If p is a point in the plane, then we denote its x- and y-coordinates by p_x and p_y, respectively. The *north-east quadrant* of p is defined as $NE(p) = [p_x, \infty) \times [p_y, \infty)$. Similarly, the *south-west quadrant* of p is defined as $SW(p) = (-\infty, p_x] \times (-\infty, p_y]$. The Manhattan distance between two points p and q is given by $d_1(p, q) = |p_x - q_x| + |p_y - q_y|$. Observe that, for $q \in NE(p)$, $d_1(p, q) = (q_x + q_y) - (p_x + p_y)$.

Definition 1 *Let S be a set of n points in the plane, let c be an integer with $1 \leq c \leq n$, and let p be a point in the plane.*

1. *Assume that $|NE(p) \cap S| \geq c$. We define $closest_c(p)$ to be the set of the c points in $NE(p) \cap S$ that are closest (with respect to d_1) to p.*
2. *Assume that $|NE(p) \cap S| < c$. We define $closest_c(p)$ to be $NE(p) \cap S$.*

The set $closest_c(p)$ can equivalently be described as follows. Consider a line with slope -1 through p. We move this line to the right until it has encountered c points of $NE(p) \cap S$ or it has encountered all points in $NE(p) \cap S$, whichever occurs first. The set $closest_c(p)$ is the subset of $NE(p) \cap S$ that are encountered during this process.

We will see in Section 3 that data structures answering the queries in (i) and (ii) above in $O(\log n)$ time, while using $O(n)$ space, can be obtained from the following result:

Theorem 4 *Let S be a set of n points in the plane and let c be an integer with $1 \leq c \leq n$. There exists a data structure of size $O(c^2 n)$ such that for any query point p, the set $closest_c(p)$ can be computed in $O(\log n + c)$ time.*

The proof of Theorem 4 will be given in Section 2. In Section 4, we will reduce range closest pair queries with squares, to range minimum weight queries, again with squares, and the queries of Section 3. Finally, in Section 5, we will present our reduction in the other direction.

2 Answering $closest_c(p)$ Queries

In this section, we will prove Theorem 4. Throughout this section, S denotes a set of n points in the plane and c denotes an integer with $1 \leq c \leq n$. We assume for simplicity that no two points in S are (i) on a vertical line, (ii) on a horizontal line, and (iii) on a line with slope -1. We will use the notion of a staircase polygon, as illustrated in Figure 1.

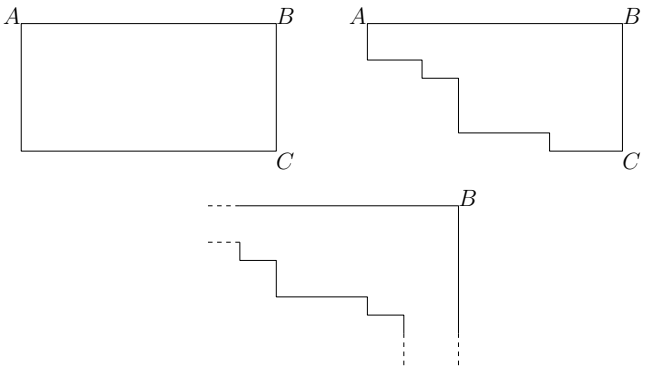


Figure 1: Staircase polygons.

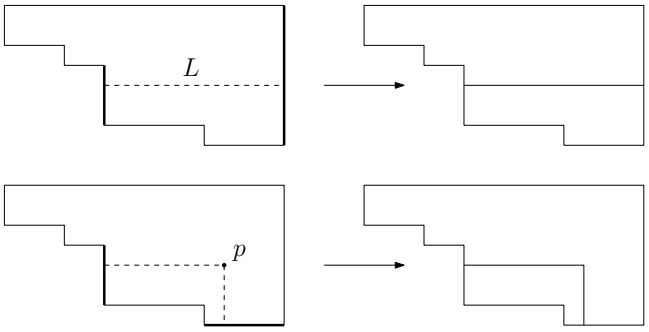


Figure 2: Illustrating Observation 1. Each thick edge is divided into two new edges.

Definition 2 (Staircase polygon) *A staircase polygon consists of (i) a horizontal edge AB, where A is to the left of B, (ii) a vertical edge CB where C is below B, and (iii) a polygonal path consisting of alternating vertical and horizontal edges, where the leftmost edge is vertical with top endpoint A and the rightmost edge is horizontal with right endpoint C.*

In the first two staircase polygons in Figure 1, the vertices A, B, and C have finite x- and y-coordinates. In the third staircase polygon, the vertex A can be thought of having an x-coordinate of $-\infty$ and the left-most edge as being infinitely far off to the left. Similarly, the vertex C has a y-coordinate of $-\infty$ and the bottom-most edge is infinitely far off in the downward direction. The vertex B may have x- and y- coordinates of ∞. In particular, the entire plane is considered a staircase polygon.

The following observation is illustrated in Figure 2.

Observation 1 *Let P be a staircase polygon.*

1. *If L is a horizontal or vertical line that intersects P, then L divides P into two staircase polygons, P_1 and P_2. The total number of edges of P_1 and P_2 (counting shared edges only once) is at most 3 more than the number of edges belonging to P.*

2. *Let p be a point in the interior of P. The boundary of $SW(p)$ divides P into two staircase polygons, P_1 and P_2. The total number of edges of P_1 and P_2 (counting shared edges only once) is at most 4 more than the number of edges belonging to P.*

2.1 Constructing the Data Structure

We order the points p in S by their p_x+p_y values and use $p^{(k)}$ to denote the k^{th} point in this ordering. Observe that this is the order in which the points of S are visited when moving a line with slope -1 from left to right.

We iteratively construct a subdivision of the plane into staircase polygons. We will refer to each such polygon as a *cell*. The 0^{th} subdivision $SD^{(0)}$ consists of one single cell, the plane itself.

In the k^{th} iteration, we add the point $p^{(k)}$ to the $(k-1)^{th}$ subdivision $SD^{(k-1)}$: From the point $p^{(k)}$, we extend a ray horizontally to the left until it has encountered c vertical edges of $SD^{(k-1)}$ or reaches $-\infty$, whichever occurs first. For $i = 1, \ldots, c-1$, the part of the ray between the i^{th} and $(i+1)^{th}$ vertical edges divides a cell of $SD^{(k-1)}$ into two cells. We also extend a ray from $p^{(k)}$ vertically downward until it has encountered c horizontal edges of $SD^{(k-1)}$ or reaches $-\infty$, whichever occurs first. For $i = 1, \ldots, c-1$, the part of the ray between the i^{th} and $(i+1)^{th}$ horizontal edges divides a cell of $SD^{(k-1)}$ into two cells. Finally, the boundary of $SW(p^{(k)})$ divides the cell of $SD^{(k-1)}$ that contains $p^{(k)}$ into two cells. The resulting subdivision is $SD^{(k)}$. The entire construction is illustrated in Figure 3.

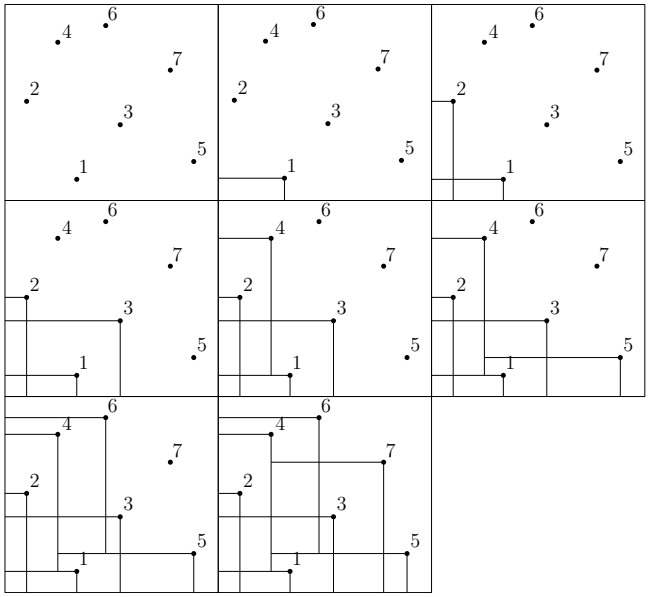


Figure 3: Constructing the sequence of subdivisions for $n = 7$ and $c = 2$.

The following lemma follows, by induction on k, from Observation 1.

Lemma 5 *For every k with $0 \le k \le n$, every cell of the subdivision $SD^{(k)}$ is a staircase polygon.*

Consider the final subdivision $SD^{(n)}$. With each cell C of this subdivision, we store the set $S_c(C) := closest_c(z)$, where z is the top-right vertex of C. Finally, we build a point location data structure for the subdivision $SD^{(n)}$; see Kirkpatrick [8]. This completes the description of the data structure.

Definition 3 *Let C be a cell in $SD^{(k)}$. The* north-east closure *of C, $NEC(C)$, consists of its interior, the topmost edge of C (without its leftmost point), and the rightmost edge of C (without its lowest point).*

For the query algorithm, consider a query point p. We first locate p in the subdivision $SD^{(n)}$, and find the (unique) cell C such that $p \in NEC(C)$. The query algorithm returns the set $S_c(C)$.

The following lemma proves the correctness of this query algorithm.

Lemma 6 *For any query point p in the plane, let C be the cell of $SD^{(n)}$ that is returned by the point location query. Then $S_c(C) = closest_c(p)$.*

A proof of Lemma 6 can be found in the Appendix.

2.2 Space Requirement and Query Time

We start by bounding the number of cells of the final subdivision $SD^{(n)}$. Clearly, $SD^{(0)}$ consists of only one cell. For each k, during the construction of the subdivision $SD^{(k)}$ from $SD^{(k-1)}$, at most $2c-1$ cells are divided into two new cells and, thus, the total number of cells increases by at most $2c-1$. It follows that the number of cells in $SD^{(n)}$ is at most $1 + n(2c-1) = O(cn)$.

Each cell C of $SD^{(n)}$ stores a set $S_c(C)$ of size at most c. Therefore, the total size of all these sets $S_c(C)$ is $O(c^2 n)$.

Next, we bound the number of edges of $SD^{(n)}$. The initial subdivision $DS^{(0)}$ is the entire plane, which we regard to be an infinite rectangle consisting of four edges. By Lemma 5, each cell in each subdivision $SD^{(k)}$ is a staircase polygon. Thus, by Observation 1, at most 4 new edges are added when such a cell is divided. Therefore, the number of edges increases by at most $4(2c-1)$ when constructing $SD^{(k)}$ from $SD^{(k-1)}$. Thus, the total number of edges in the final subdivision $SD^{(n)}$ is at most $4 + n \cdot 4(2c-1) = O(cn)$. It follows that the point location data structure uses $O(cn)$ space.

We have shown that the space used by the entire data structure is $O(c^2 n)$.

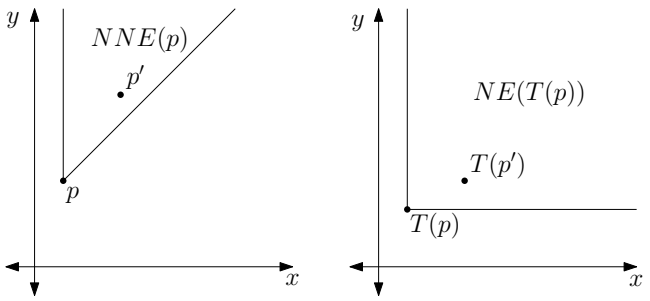


Figure 4: T transforms $NNE(p)$ into $NE(T(p))$.

The query algorithm, with query point p, first performs point location, which takes $O(\log(cn)) = O(\log n)$ time, because $c \leq n$. Reporting the set $closest_c(p)$ takes $O(c)$ time. Thus, the total query time is $O(\log n + c)$.

This completes the proof of Theorem 4.

3 Some Related Queries

In this section, we use the data structure of Theorem 4 to solve several related query problems.

Definition 4 *Let p be a point in the plane and consider the line with slope 1 through p. This line divides $NE(p)$ into two cones, each one having an angle of 45°. We denote the upper cone by $NNE(p)$ and the lower cone by $ENE(p)$.*

Lemma 7 *Let S be a set of n points in the plane and let c be an integer with $1 \leq c \leq n$. There exists a data structure of size $O(c^2 n)$ which can perform the following query in $O(\log n + c)$ time: Given a query point p, find the smallest square that has p as its bottom-left corner and contains c points of S.*

Proof. Assume we know the set L_1 consisting of the c lowest points of $NNE(p) \cap S$ and the set L_2 consisting of the c leftmost points of $ENE(p) \cap S$. Then we obtain the answer to the query in $O(c)$ time by selecting the c^{th} smallest element in the sequence $d_\infty(p, q)$, $q \in L_1 \cup L_2$, where $d_\infty(p, q) = \max\{|p_x - q_x|, |p_y - q_y|\}$.

We will describe how the data structure of Theorem 4 can be used to find the set L_1 in $O(\log n + c)$ time. Finding the set L_2 can be done in a symmetric way.

Consider the transformation T that maps any point $q = (q_x, q_y)$ to the point $T(q) = (q_x, q_y - q_x)$. We compute the set $S' = \{T(q) : q \in S\}$ and construct the data structure of Theorem 4 for S'.

Observe that $p' \in NNE(p)$ if and only if $T(p') \in NE(T(p))$; refer to Figure 4. Furthermore, if $p' \in NNE(p)$, then $d_1(T(p), T(p')) = d_1((p_x, p_y - p_x), (p'_x, p'_y - p'_x)) = (p'_x + (p'_y - p'_x)) - (p_x + (p_y - p_x)) = p'_y - p_y$. Thus, p' is one of the c lowest points in $NNE(p) \cap S$ if and only if $T(p')$ is one of the c points in $NE(T(p)) \cap S'$ that is closest (with respect to d_1) to $T(p)$.

Thus, for a given query point p, by querying the data structure for S' with $T(p)$, we obtain the set L_1. By Theorem 4, the amount of space used is $O(c^2 n)$ and the query time is $O(\log n + c)$. □

Lemma 8 *Let S be a set of n points in the plane and let c be an integer with $0 \leq c \leq n - 1$. There exists a data structure of size $O(c^2 n)$ which can perform the following query in $O(\log n + c)$ time: Given a query square R, decide whether $|R \cap S| \leq c$, and if so, report the points of $R \cap S$.*

Proof. We store the set S in the data structure of Lemma 7, with c replaced by $c + 1$.

Let p be the bottom-left corner of the query square R. By querying the data structure, we obtain the smallest square R' that has p as its bottom-left corner and contains $c + 1$ points of S. It is clear that one of these $c + 1$ points is on the top or right edge of R'; let this point be p'.

If $p' \notin R$ then R is properly contained in R' and, thus, $|R \cap S| \leq c$. In this case, since $R \cap S \subset (R' \cap S)$, the points of $R \cap S$ can be reported in $O(c)$ time.

If $p' \in R$ then $|R \cap S| > c$. This fact is reported. □

4 From Minimum Weight Queries to Closest-Pair Queries

In this section, we prove Theorem 1. Let S be a set of n points in the plane.

We assume that, for any set V of m weighted points in the plane, we can construct a data structure $DS_{MW}(V)$ that can report, for any query square R, the minimum weight of any point in $R \cap V$. We denote the space and query time of this data structure by $M(m)$ and $Q(m)$, respectively. We assume that both functions M and Q are smooth, $M(m) \geq m$, and $Q(m) = \Omega(\log m)$.

We will show that DS_{MW} and the results from the previous sections can be used to obtain a data structure that supports range closest pair queries on S for ranges that are squares.

Let R be a query square. Bae and Smid [2] have shown that, in order to obtain the closest pair in $R \cap S$, the following steps are sufficient:

1. Decide if $|R \cap S| \leq 16$. If this is the case, report the points in $R \cap S$.

   Thus, we store the points of S in the data structure of Lemma 8, where $c = 16$. This uses $O(n)$ space and supports this query in $O(\log n)$ time.

2. Let p be the bottom-left corner of R. Find the smallest square that has p as its bottom-left corner and contains 5 points of S.

Thus, we store the points of S in the data structure of Lemma 7, where $c = 5$. This uses $O(n)$ space and supports this query in $O(\log n)$ time.

3. Three queries that are symmetric to 2., with p being the bottom-right, top-right, and top-left corner of R, respectively.

   Thus, we store the points of S in each of these three variants of the data structure of Lemma 7, where $c = 5$. This uses $O(n)$ space and supports these three queries in $O(\log n)$ time.

4. During preprocessing, we obtain four subsets $S_1, \ldots, S_4$ of S; these subsets may overlap. For each $k = 1, 2, 3, 4$, each point of S_k has a positive weight.

   To answer the closest pair query for R, the previous three steps give four squares $B_1, \ldots, B_4$. For each $k = 1, \ldots, 4$, we find the minimum weight of any point in $B_k \cap S_k$.

   Thus, for each $k = 1, \ldots, 4$, we store the weighted point set S_k in the data structure $DS_{MW}(S_k)$. Since S_k has size at most n, this uses $O(M(n))$ space and supports these four queries in $O(Q(n))$ time.

5. The previous four steps give four squares $C_1, \ldots, C_k$, each containing at most 5 points of S. For each $k = 1, \ldots, 4$, we find the points of $C_k \cap S$.

   Thus, we store the points of S in the data structure of Lemma 8, where $c = 5$. This uses $O(n)$ space and supports this query in $O(\log n)$ time.

6. The results of the queries in these five steps give us sufficient information to compute the closest pair in $R \cap S$ in $O(1)$ time.

To conclude, the total amount of space used is $O(M(n) + n) = O(M(n))$ and the total query time is $O(Q(n) + \log n) = O(Q(n))$. This proves Theorem 1.

5 From Closest-Pair Queries to Minimum Weight Queries

In this final section, we prove Theorem 2. Let S be a set of n weighted points in the plane. For each point p in S, we denote its weight by $\omega(p)$.

We assume that, for any set V of m points in the plane, we can construct a data structure $DS_{CP}(V)$ that can report, for any query square R, the closest pair in $R \cap V$. We denote the space and query time of this data structure by $M(m)$ and $Q(m)$, respectively. We assume that both functions M and Q are smooth, $M(m) \geq m$, and $Q(m) = \Omega(\log m)$.

We will show that DS_{CP} and the data structure of Lemma 8 can be used to obtain a data structure that supports range minimum weight queries on S for ranges that are squares.

We may assume, without loss of generality, that all weights $\omega(p)$ are positive, pairwise distinct, and strictly less than 1. (If this is not the case, then we sort the sequence of weights, breaking ties arbitrarily, and replace each weight by $1/(2n)$ times its position in the sorted order.)

Let δ be the closest pair distance in the set S. For each point p in S, define the points

$$p^+ = (p_x + \delta \cdot \omega(p)/3, p_y)$$

and

$$p^- = (p_x - \delta \cdot \omega(p)/3, p_y)\,,$$

and let $S' = \{p^+ : p \in S\} \cup \{p^- : p \in S\}$.

Our data structure for minimum weight queries consists of the following:

1. We store the points of S in the data structure of Lemma 8, where $c = 1$.

2. We store the points of $S \cup S'$ in the data structure $DS_{CP}(S \cup S')$.

The query algorithm is as follows. Let R be a query square. First, we decide whether $|R \cap S| \leq 1$. If this is the case, then we obtain the set $R \cap S$. If this set contains one point, say p, then we return $\omega(p)$; otherwise, we return the fact that $R \cap S$ is empty.

Assume that $|R \cap S| \geq 2$. Then we query $DS_{CP}(S \cup S')$ for the closest pair in $R \cap (S \cup S')$. Let (p, a) be this closest pair. In Lemma 11, we will prove that $p \in R \cap S$ and $a \in R \cap \{p^+, p^-\}$. We return $\omega(p)$.

Since $|S| = n$ and $|S'| = 2n$, the total amount of space used by the data structure is $O(n) + M(3n) = O(M(n))$ and the total query time is $O(\log n) + Q(3n) = O(Q(n))$.

To complete the proof of Theorem 2, it remains to prove the correctness of the query algorithm. We will present this proof in the next subsection.

5.1 Correctness of the Query Algorithm

We denote the Euclidean distance between two points a and b by $d(a, b)$. We start with two preliminary lemmas.

Lemma 9 *Let R be a square such that $|R \cap S| \geq 2$. Then for each point p in $R \cap S$, at least one of the points p^+ and p^- is in R.*

Proof. Let ℓ be the side length of R. The distance between any two distinct points of $R \cap S$ is at least δ and at most $\ell \cdot \sqrt{2}$. It follows that $\delta \leq \ell \cdot \sqrt{2}$.

Let p be an arbitrary point in $R \cap S$. We may assume, without loss of generality, that p is in the left half of R, i.e., the distance between p and the right boundary of R is at least $\ell/2$. Since $\omega(p) < 1$,

$$d(p, p^+) = \delta \cdot \omega(p)/3 < \delta/3 < \ell/2$$

and, thus, the point p^+ is in R. □

Lemma 10 *Let p and q be two distinct points in S, and let $a \in \{p^+, p^-\}$ and $b \in \{q^+, q^-\}$. Then the following inequalities hold:*

1. *Both $d(p,a)$ and $d(q,b)$ are less than $\delta/3$.*
2. *$d(p,q) \geq \delta$.*
3. *Both $d(p,b)$ and $d(a,q)$ are larger than $2\delta/3$.*
4. *$d(a,b) > \delta/3$.*

Proof. Recall that the weights of all points in S are less than 1. Since $d(p,a) = \delta \cdot \omega(p)/3 < \delta/3$ and $d(q,b) = \delta \cdot \omega(q)/3 < \delta/3$, the first claim holds. The second claim follows from the definition of δ. The third claim holds because

$$\delta \leq d(p,q) \leq d(p,b) + d(b,q) < d(p,b) + \delta/3$$

and

$$\delta \leq d(p,q) \leq d(p,a) + d(a,q) < \delta/3 + d(a,q).$$

The fourth claim holds because

$$\delta \leq d(p,q) \leq d(p,a)+d(a,b)+d(b,q) < \delta/3+d(a,b)+\delta/3.$$

□

The next lemma states that the output of the query in $DS_{CP}(S \cup S')$ consists of one point p in S and one point in $\{p^+, p^-\}$.

Lemma 11 *Let R be a square such that $|R \cap S| \geq 2$. The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (p,a), for some $p \in R \cap S$ and $a \in R \cap \{p^+, p^-\}$.*

Proof. We consider the three possible cases, depending on whether the closest pair distance in $R \cap (S \cup S')$ is attained by two points of S (Case 1), two points of S' (Case 2), or one point of S and one point of S' (Case 3). As we will see, neither of the first two cases can happen.

Case 1: The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (p,q), where p and q are distinct points in $R \cap S$.

By Lemma 9, there exist points $a \in \{p^+, p^-\}$ and $b \in \{q^+, q^-\}$, such that both a and b are in R. Therefore, the closest pair distance in $R \cap (S \cup S')$ is at most the closest pair distance in $\{p, q, a, b\}$, which, by Lemma 10, is less than $d(p,q)$. This is a contradiction. Thus, this case cannot happen.

Case 2: The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (a,b), where a and b are distinct points in $R \cap S'$.

Let p and q be the points in S such that $a \in \{p^+, p^-\}$ and $b \in \{q^+, q^-\}$. Note that p or q may be outside R.

First assume that $p = q$. Then, $\{a,b\} = \{p^+, p^-\}$ and, thus, $p \in R$. But then $d(p,a) < d(a,b)$, which is a contradiction.

Thus, $p \neq q$. By Lemma 10, $d(a,b) > \delta/3$. Let r be the point in $R \cap S$ whose weight is minimum. By Lemma 9, there exists a point $c \in \{r^+, r^-\}$, such that c is in R, and, by Lemma 10, $d(r,c) < \delta/3$. It follows that $d(r,c) < d(a,b)$, which is a contradiction. Thus, Case 2 cannot happen.

Case 3: The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (a,q), where a is a point in $R \cap S'$ and q is a point in $R \cap S$.

Let p be the point in S such that $a \in \{p^+, p^-\}$. The claim in the lemma follows if we can show that $p = q$.

Assume that $p \neq q$. By Lemma 9, there exists a point $b \in \{q^+, q^-\}$, such that b is in R. We obtain a contradiction, because, by Lemma 10, $d(q,b) < \delta/3$ and $d(a,q) > 2\delta/3$. □

The next lemma will complete the correctness proof of our query algorithm.

Lemma 12 *Let R be a square such that $|R \cap S| \geq 2$. Let p be a point in $R \cap S$ and let a be a point in $\{p^+, p^-\}$, such that the closest pair distance in $R \cap (S \cup S')$ is attained by (p,a). (By Lemma 11, p and a exist.) Then the minimum weight of any point in $R \cap S$ is equal to $\omega(p)$.*

Proof. Let q be the point in $R \cap S$ whose weight is minimum. By Lemma 9, there exists a point $b \in \{q^+, q^-\}$, such that b is in R. If $q \neq p$, then

$$d(q,b) = \delta \cdot \omega(q)/3 < \delta \cdot \omega(p)/3 = d(p,a),$$

which is a contradiction. Thus, $q = p$. □

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. *Computational Geometry: Theory and Applications*, 46:631–639, 2013.

[2] S. W. Bae and M. Smid. Closest-pair queries in fat rectangles. *Computational Geometry: Theory and Applications*, 83:1–8, 2019.

[3] T. M. Chan, S. Rahul, and J. Xue. Range closest-pair search in higher dimensions. In *Proceedings of the 16th Algorithms and Data Structures Symposium*, volume 11646 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2019.

[4] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17:427–462, 1988.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008.

[6] P. Gupta. Algorithms for range-aggregate query problems involving geometric aggregation operations. In *Proceedings of the 16th Annual International Symposium on Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 892–901. Springer, 2005.

[7] P. Gupta, R. Janardan, Y. Kumar, and M. Smid. Data structures for range-aggregate extent queries. *Computational Geometry: Theory and Applications*, 47:329–347, 2014.

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

[9] J. Shan, D. Zhang, and B. Salzberg. On spatial-range closest-pair query. In *Proceedings of the 8th International Symposium on Spatial and Temporal Databases*, volume 2750 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2003.

[10] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. Technical Report IIIT/TR/2007/80, International Institute of Information Technology Hyderabad, 2007.

[11] J. Xue, Y. Li, and R. Janardan. Approximate range closest-pair search. In *Proceedings of the 30th Canadian Conference on Computational Geometry*, pages 282–287, 2018.

[12] J. Xue, Y. Li, S. Rahul, and R. Janardan. New bounds for range closest-pair problems. In *Proceedings of the 34th International Symposium on Computational Geometry*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 73:1–73:14, 2018.

Appendix

We state a few definitions and observations in preparation for proving Lemma 6. As in Section 2.1, S is a set of n points ordered by their $p_x + p_y$ values, $p^{(k)}$ is the k^{th} point in this ordering, and $1 \leq c \leq n$.

Definition 5 *$S^{(k)}$ is the set of the first k points of S, that is, $S^{(k)} = \{p^{(1)}, \ldots, p^{(k)}\}$. Note that $S^{(n)} = S$.*

Definition 6 *For any cell $C \in SD^{(k)}$, the depth of that cell is $depth(C) = |NE(z) \cap S^{(k)}|$, where z is the top-right vertex of the cell.*

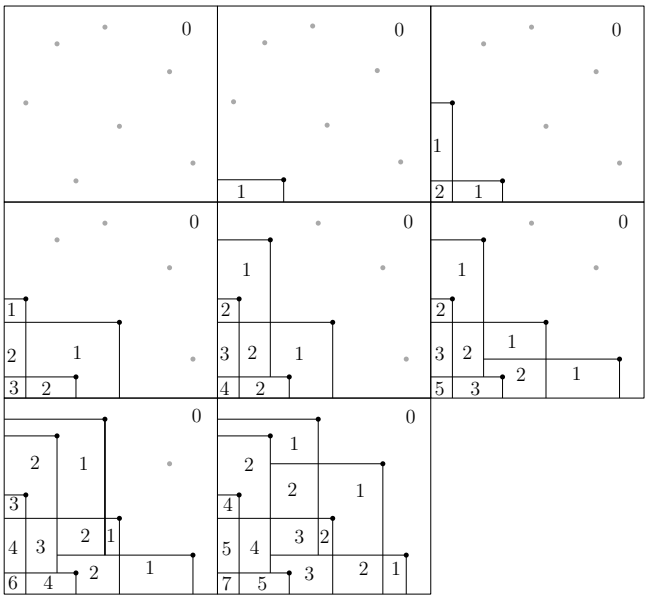


Figure 5: Constructing the sequence of subdivisions for $n = 7$ and $c = 2$, with the depth of each cell displayed inside it.

The following observation is illustrated in Figure 5.

Observation 2 *For all k with $0 \leq k \leq n$, there is exactly one cell of depth 0 in $SD^{(k)}$, and $p^{(k)}$ belongs to the cell of depth 0 in $SD^{(k-1)}$. If L is a horizontal or vertical ray starting at $p^{(k)}$ and moving left or down respectively, the first c cells encountered by L in $SD^{(k-1)}$ have depths of $0, 1, \ldots, c-1$, and every cell afterwards has a depth of at least c. In particular, if $1 \leq c_1 \leq c-1$, the unique cell of depth c_1 that intersects L will be split into two cells of $SD^{(k)}$ by the part of the L between the c_1^{th} and $(c_1+1)^{th}$ edges encountered.*

Definition 7 *Let p be a point in the plane.*

1. *Assume that $|NE(p) \cap S^{(k)}| \geq c$. We define $closest_c^{(k)}(p)$ to be the set of the c points in $NE(p) \cap S^{(k)}$ that are closest (with respect to d_1) to p.*

2. *Assume that* $|NE(p) \cap S^{(k)}| < c$. *We define* $closest_c^{(k)}(p)$ *to be* $NE(p) \cap S^{(k)}$.

3. *If* C *is a cell in* $SD^{(k)}$, *then* $S_c^{(k)}(C) := closest_c^{(k)}(z)$ *where* z *is the top-right vertex of* C.

Observation 3 *If* p *is any point in the plane and* $p^{(i)}, p^{(j)} \in NE(p)$, *where* $i < j$, *then since* $p_x^{(i)} + p_y^{(i)} < p_x^{(j)} + p_y^{(j)}$, *we have* $d_1(p, p^{(i)}) < d_1(p, p^{(j)})$. *Thus, the* set of c points closest to p in $S^{(k)} \cap NE(p)$ *in the definition of* $closest_c^{(k)}(p)$ *is the same as the* set of c points of lowest order in $S^{(k)} \cap NE(p)$. *It also follows that if* $NE(p^1) \cap S^{(k_1)} = NE(p^2) \cap S^{(k_2)}$, *then* $closest_c^{(k_1)}(p^1) = closest_c^{(k_2)}(p^2)$

Lemma 13 *Let* k *be any integer with* $0 \leq k \leq n$ *and let* p^1 *and* p^2 *be any points in the plane which belong to the northeast closure of the same cell in* $SD^{(k)}$, *and* $|S^{(k-1)} \cap NE(p^1)| < c$. *Then* $p^{(k)} \in NE(p^1)$ *if and only if* $p^{(k)} \in NE(p^2)$.

Proof. Note that p^1 and p^2 must have belonged to the northeast closure of the same cell in $SD^{(k-1)}$, so there exists a cell $C \in SD^{(k-1)}$ such that $p^1, p^2 \in NEC(C)$. Let z be the top-right vertex of C. Then since $NE(z) \subseteq NE(p^1)$, we have $S^{(k-1)} \cap NE(z) \subseteq S^{(k-1)} \cap NE(p^1)$, so $depth(C) = |S^{(k-1)} \cap NE(z)| < c$.

We prove that $p^{(k)} \in NE(p^1)$ implies $p^{(k)} \in NE(p^2)$. The converse is symmetric.

Let $p^{(k)} \in NE(p^1)$ and suppose $p^{(k)} \notin NE(p^2)$.

If $depth(C) = 0$, then since $p^1 \in SW(p^{(k)})$ and $p^2 \notin SW(p^{(k)})$, p^1 and p^2 will be in the northeast closure of different cells in $SD^{(k)}$, contradicting the fact that $p^1, p^2 \in NEC(C)$.

Now suppose $1 \leq depth(C) \leq c - 1$. Since $p^{(k)} \notin NE(p^2)$, $p^{(k)}$ is strictly below or strictly to the left of p^2; without loss of generality, we assume the former. Since $p^{(k)} \in NE(p^1)$, $p^{(k)}$ is above or at the same height as p^1. Thus, the horizontal ray starting at $p^{(k)}$ and moving left will encounter C, and since $1 \leq depth(C) \leq c - 1$, by Observation 2, C will be split into two new cells of $SD^{(k)}$. p^1 will be in the northeast closure of the lower cell and p^2 will be in the northeast closure of the upper cell, again contradicting the fact that $p^1, p^2 \in NEC(C)$. □

The following lemma implies Lemma 6 when $k = n$.

Lemma 14 *For any* k *with* $0 \leq k \leq n$ *and for any point* p *in the plane, let* C *be the cell of* $SD^{(k)}$ *such that* $p \in NEC(C)$. *Then* $S_c^{(k)}(C) = closest_c^{(k)}(p)$.

Proof. We use induction on k.

When $k = 0$, $S^{(0)} = \emptyset$, so the claim clearly holds. Now let $k \geq 1$ and suppose that for all points p, if $p \in NEC(C)$ where $C \in SD^{(k-1)}$, then $S_c^{(k-1)}(C) = closest_c^{(k-1)}(p)$. Let p be any point in the plane, let C be the cell in $SD^{(k)}$ such that $p \in NEC(C)$, and let z be the top-right vertex of C. We must show $closest_c^{(k)}(z) = S_c^{(k)}(C) = closest_c^{(k)}(p)$. Note that $z \in NEC(C)$ and so p and z must have belonged to the northeast closure of the same cell in $SD^{(k-1)}$. Thus, by hypothesis, $closest_c^{(k-1)}(p) = closest_c^{(k-1)}(z)$.

We consider two cases based on the cardinality of $S^{(k-1)} \cap NE(p)$

For the first case, suppose $|S^{(k-1)} \cap NE(p)| \geq c$.

Then $closest_c^{(k-1)}(p) = \{p^{(i_1)}, \ldots, p^{(i_c)}\} = closest_c^{(k-1)}(z)$. If $p^{(k)} \notin NE(p)$, then $S^{(k)} \cap NE(p) = S^{(k-1)} \cap NE(p)$, so $closest_c^{(k)}(p) = closest_c^{(k-1)}(p)$. If $p^{(k)} \in NE(p)$, then since $i_1, \ldots, i_c < k$, $p^{(i_1)}, \ldots, p^{(i_c)}$ are still the c points of lowest order in $S^{(k)} \cap NE(p)$, so again, $closest_c^{(k)}(p) = closest_c^{(k-1)}(p)$. Similarly, it can be shown that $closest_c^{(k)}(z) = closest_c^{(k-1)}(z)$. Thus, $closest_c^{(k)}(p) = closest_c^{(k-1)}(p) = closest_c^{(k-1)}(z) = closest_c^{(k)}(z)$.

For the second case, suppose $|S^{(k-1)} \cap NE(p)| < c$.

Since p and z belong to the northeast closure of the same cell in $SD^{(k)}$, by Lemma 13, $p^{(k)} \in NE(p)$ if and only if $p^{(k)} \in NE(z)$. If $p^{(k)} \in NE(p)$, then $p^{(k)} \in NE(z)$ and so $\{p^{(k)}\} \cap NE(p) = \{p^{(k)}\} = \{p^{(k)}\} \cap NE(z)$. If $p^{(k)} \notin NE(p)$, then $p^{(k)} \notin NE(z)$ and so $\{p^{(k)}\} \cap NE(p) = \emptyset = \{p^{(k)}\} \cap NE(z)$. Thus, $\{p^{(k)}\} \cap NE(p) = \{p^{(k)}\} \cap NE(z)$.

Now since $|S^{(k-1)} \cap NE(p)| < c$, $closest_c^{(k-1)}(p) = S^{(k-1)} \cap NE(p)$. Since $closest_c^{(k-1)}(p) = closest_c^{(k-1)}(z)$, $|closest_c^{(k-1)}(z)| < c$ so it must be that $|S^{(k-1)} \cap NE(z)| < c$ and $closest_c^{(k-1)}(z) = S^{(k-1)} \cap NE(z)$. Then $S^{(k)} \cap NE(p) = (S^{(k-1)} \cap NE(p)) \cup (\{p^{(k)}\} \cap NE(p)) = (closest_c^{(k-1)}(p)) \cup (\{p^{(k)}\} \cap NE(p)) = (closest_c^{(k-1)}(z)) \cup (\{p^{(k)}\} \cap NE(z)) = (S^{(k-1)} \cap NE(z)) \cup (\{p^{(k)}\} \cap NE(z)) = S^{(k)} \cap NE(z)$. Thus, by Observation 3, $closest_c^{(k)}(p) = closest_c^{(k)}(z)$. □

Parallel Topological Sweep

Ming Ouyang*

Abstract

On input of a line arrangement, topological sweep outputs the line intersections in a topological order. The intersection of two lines is *ready* if all intersections to the left on these two lines have been processed. The classical algorithm processes the ready intersections one at a time. This article describes the first attempt to process the ready intersections in parallel. It is proved that, at the beginning of the sweep of a random arrangement, the expected number of ready intersections is a constant fraction of the number of lines. After the first batch, empirical data show that many intersections become ready batch after batch. Two new implementations are described. On arrangements of 300,000 lines, a new serial implementation is 3.92 times the speed of a serial implementation in the literature, and the first parallel implementation is 4.2 times the speed of the new one.

1 Introduction

Topological sweep [3] is a classical algorithm in computational geometry. The input is an arrangement of n lines in the plane. The intersection of two lines is a *vertex*. An arrangement is *simple* if any two lines intersect at a vertex, but no three do so. The algorithm sweeps the arrangement — reporting the vertices — using $O(n^2)$ time, which is asymptotically optimal. It is used in efficient algorithms for applications, such as data depth [4, 5, 7, 8, 9]. The algorithm is implemented in C by Rosenberger [3, 12] and in `C++` using the LEDA library by Miller *et al.* [8]. It is extended to handle non-simple arrangements by Rafalin *et al.* [11] — they implement the extended method in `C++` without using any standard libraries. The algorithm and implementations are serial in nature. Parallel topological sweep is needed for two reasons. First, in the past five decades, the performance of computers has more or less doubled every 18 months. This so-called Moore's Law, however, is showing signs of plateauing. Second, experimental scientists, enabled by technology, are collecting more and larger data sets than ever. Analysis of large data sets, such as finding the Tukey median [8], is difficult without parallelization. The author is unaware of any prior attempt at parallelizing topological sweep.

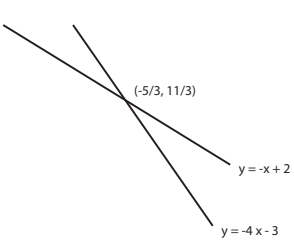
Section 2 reviews the line-point duality and the classical algorithm. Section 3 examines how to parallelize it. Section 4 studies the expected concurrency in random arrangements. It is proved that $\Omega(n)$ intersections are ready at the beginning of the sweep. Empirical data show that, on average, a constant fraction of the lines are engaged in ready pairs throughout the sweep. Section 5 describes a new serial implementation in C and the first parallel implementation in C and OpenMP. The new serial code is 3.92 times the speed of the Rafalin code [11]. The parallel code is 4.2 times the speed of the new serial code and more than 16 times that of the Rafalin code. Section 6 concludes with discussion.

2 Serial Topological Sweep

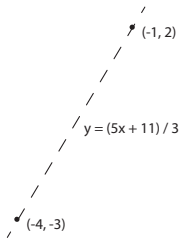
The description of the classical algorithm is expanded beyond that of [3] to include the dual space of point arrangement, which makes easy a proof in Section 4. Let A be a simple arrangement of n lines. As in [3], it is assumed that none of the lines is vertical. Some applications, such as data depth [5, 8], need to process the vertices of A in some order. They require that the vertices on the same line be listed monotonically — vertices on different lines may come in any order. Topological sweep produces such a topological sort of the vertices.

The line-point duality maps a line $y = c_1x + c_0$ to the point (c_1, c_0), and a point (c_1, c_0) to the line $y = -c_1x + c_0$. Fig. 1(a) shows two lines and their intersection. Fig. 1(b) is the dual arrangement. The duality preserves incidence — the dual line of the intersection point is incident upon the dual points of the lines. Sweeping a line can be construed as walking along its upper and lower sides to detect whether the line comes to an intersection as the upper or lower line. Fig. 1(c) shows walking the lines of Fig. 1(a) from left to right. For the upper line, $y = -4x - 3$, its lower sidewalk is blocked by the lower line, but its upper sidewalk overpasses the intersection and continues to the right. For the lower line, $y = -x + 2$, its upper sidewalk is blocked by the upper line, but its lower sidewalk underpasses the intersection and continues to the right. The duals of the upper and lower sidewalks are, metaphorically, the left and right halves, respectively, of the dual point. Imagine the dual point as a clock. The dual of the upper sidewalk — moving from negative infinity to infinity — is a hand, the lower hand, that rotates from six o'clock to twelve o'clock, whereas the dual of the lower sidewalk is the upper hand that rotates from twelve o'clock

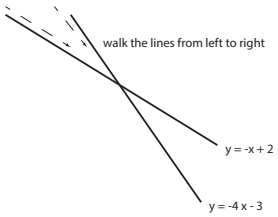
*Department of Computer Science, UMass Boston



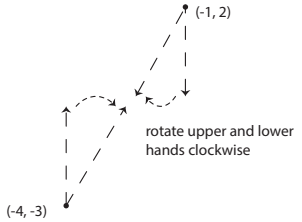
(a) $y = -4x-3$ and $y = -x+2$ intersect at $(-5/3, 11/3)$.



(b) $y = (5x+11)/3$ is incident on $(-4,-3)$ and $(-1,2)$.

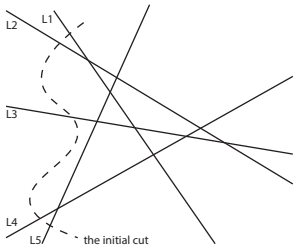


(c) Walk the lower side of $y = -4x - 3$ and the upper side of $y = -x + 2$.

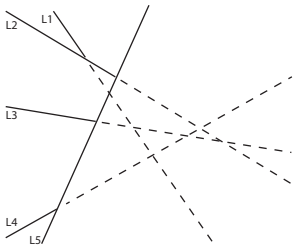


(d) Rotate upper hand of $(-4,-3)$ and lower hand of $(-1,2)$ clockwise.

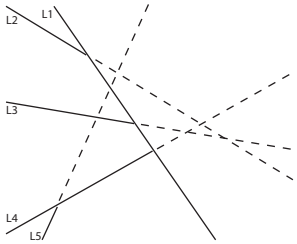
Figure 1: Line-point duality.



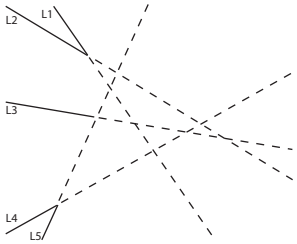
(a) The initial cut.



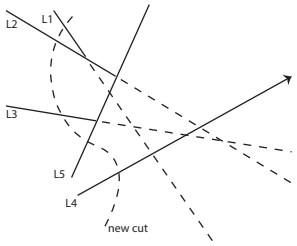
(b) Solid lines: initial UHT.



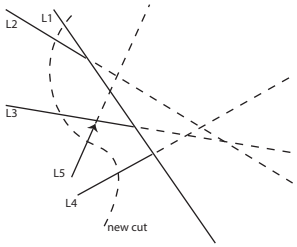
(c) Solid lines: initial LHT.



(d) Set-intersection of horizon trees.



(e) Processing (L_4, L_5) and updating the UHT.



(f) Processing (L_4, L_5) and updating the LHT.

Figure 2: An arrangement of five lines.

to six o'clock. During such a rotation, a hand must make a stop and point at any other point. When the upper hand of one point lines up with the lower hand of another point, the two hands merge and form the line dual of the primal vertex. As shown in Fig. 1(d), the upper hand of $(-4,-3)$ rotates from twelve o'clock until it points at $(-1,2)$. The lower hand of $(-1,2)$ rotates from six o'clock until it points at $(-4,-3)$. Not shown in Fig. 1(d), both the lower hand of $(-4,-3)$ and the upper hand of $(-1,2)$ make a half-circle rotation.

The lines in the arrangement are sorted by their slopes. A topological sweep line — a *cut* — is monotonic in the y-direction, intersects each of the n lines once, and does not pass through any vertices. Fig. 2(a) shows the first cut of an arrangement of five lines, $L_1, \ldots, L_5$. The first cut walks the lines from left to right and stops before the vertices. An array, `cut[]`, stores the order of the lines along the cut. Initially, `cut[1]` is L_1, `cut[2]` is L_2, and so on. When the cut advances over a vertex, the two intersecting lines, which must have been adjacent in the cut, will swap their places.

The *upper and lower horizon trees* — UHT and LHT — are the main data structures. The solid lines in Fig. 2(b) and (c) are the initial UHT and LHT, respectively. The exposition of [3] illustrates the algorithm with the UHT. This article uses the LHT. When two lines meet in the LHT, the upper line has higher precedence — it continues to the right and blocks the lower line from proceeding. The precedence in the UHT is reversed. The trees are stored in two arrays, `uht[]` and `lht[]`, where each element is the entity that blocks the line from proceeding. For example, in the LHT, L_2, L_3, and L_4 are blocked by L_1. Each line L_i has two obstacles, `uht[`i`]` and `lht[`i`]`. The obstacle that is closer to the cut is stored in `closer[`i`]`. This array `closer[]` is a succinct representation of the set-intersection of the UHT and LHT (Fig. 2(d)). The crucial observation is that, when `closer[cut[`i`]]` is `cut[`i`+1]` and `closer[cut[`i`+1]]` is `cut[`i`]`, the intersection of `cut[`i`]` and `cut[`i`+1]` is *ready* for the sweep line to cross. In Fig. 2(d), (L_1, L_2) is a ready pair, so is (L_4, L_5). Table 1 lists the initial values of the data structures. These data structures are simplified from those in [3], which store both the left and right endpoints of the UHT and LHT. Herein only the right endpoints are kept. The simplified version is sufficient to produce a topological sort of the vertices and conducive to fast implementation.

The ready pairs are stored in an array `ready[]`. A pair is represented by the rank in the cut of its first member. In Table 1, `ready[1]` is 1 and `ready[2]` is 4. During the sweep, one ready pair is removed from `ready[]`, and up to two new pairs may be added back to it. This array `ready[]` can be managed either as a queue or a stack, because the pairs can be processed in any order. Herein lies the source of concurrency.

The *upper and lower* horizon trees correspond to the

Table 1: The initial data structures for the arrangement in Fig. 2

i	cut[i]	uht[i]	lht[i]	closer[i]	
1	1	2	∞	2	ready
2	2	5	1	1	
3	3	5	1	5	
4	4	5	1	5	ready
5	5	∞	4	4	

lower and upper sidewalks, respectively, which in turn correspond to the *upper and lower* hands, respectively, in the dual space. For example, in the LHT, L_1 goes all the way to the right, so its lower hand in the dual space rotates nonstop from six o'clock to twelve o'clock. The lower hands of L_2, L_3, and L_4 stop at the dual of L_1, and that of L_5 stops at the dual of L_4. This correspondence will be used in Section 4.

```
1  lht[1] = ∞                          //initialize LHT
       with L_1
2  for i from 2 to n                   //insert L_i into LHT
3      j = cut[i − 1]
4      currentX = ∞
5      memo = ∞
6      while (j ≥ 1 and j < ∞)         //traverse the bay
7          nextX = x−coord of intersection of L_i, L_j
8          if (nextX < currentX)
9              currentX = nextX
10             memo = j
11             j = lht[j]              //new L_j blocks old L_j
12         else
13             break                   //break out the while loop
14     lht[i] = memo
```

Listing 1: Pseudo-code of constructing the initial LHT

Listing 1 is the pseudo-code for constructing the initial LHT, which is initialized with L_1 by setting `lht[1]` to infinity. The major step is *the clockwise traversal of the bay*. Consider the scenario when L_3 enters the scene after L_1 and L_2 are already in place (Fig. 2(c)). The *bay* here consists of, *clockwise*, L_2 and L_1 — see Fig. 3(a) for a more complicated example, where Bay_3 consists of L_5, L_4, L_3, and L_1. The `while` loop on lines 6–13 performs the bay traversal and finds the next intersection of L_3, which is with L_1. Before L_3 is inserted, the bay consists of L_2 and L_1. After L_3 is inserted, L_2 drops below the horizon, and the bay consists of L_3 and L_1. L_2 is no longer visible to subsequent L_j's. The same situation of dropping below the horizon happens to L_3 after L_4 is inserted. The time of constructing the LHT is $O(n)$ because the `while` loop will iterate at most $3(n-1)$ times, accounted for in three categories. First, there is one iteration per L_i as the first iteration of the loop, for $n-1$ iterations in total. Second, there is one iteration every time when the `if` condition (line 8) is true, for at most $n-1$ iterations in total. When this happens, a previous line drops below the horizon. This can happen at most once for every line. Third, there is one iteration every time when the `if` condition (line 8) is false, for at most $n-1$ iterations in total. When this happens, the loop terminates via the `break` statement.

When the cut advances beyond a ready pair, the algorithm needs to update the horizon trees. This is illustrated with the ready pair (L_4, L_5) in Fig. 2(d). The new UHT is shown in Fig. 2(e). The right endpoint of L_5 remains the same. As for L_4, the bay beyond L_5 is empty, so `uht[4]` is set to infinity. The new LHT is shown in Fig. 2(f). The right endpoint of L_4 remains the same. As for L_5, the bay beyond L_4 — consisting of L_1 and L_3 — becomes visible. Thus, L_5 has to traverse the bay clockwise to find its new endpoint. Listing 2 is the pseudo-code for updating the LHT. The second member of the pair — the lower line — is designated as L_i. The bay traversal is performed by the `while` loop on lines 7–15. This traversal is similar to the one in the initial construction of the LHT. The complication is the `if` statement on lines 10–12. When the `if` condition is true, L_j is located to the right of the cut, and its intersection with L_i is duly considered. When it is false, the intersection of L_i and L_j is to the *left* of the cut and has already been processed, so the `if` statement has no `else` part. After the UHT and LHT are updated, the intersecting lines swap their places in the array `cut[]` and may be engaged in up to two new ready pairs.

```
1  //update LHT for ready pair (cut[r], cut[r+1])
2  i = cut[r + 1]
3  memo = ∞
4  if (r − 1 ≥ 1)
5      j = cut[r − 1]
6      currentX = ∞
7      while (j ≥ 1 and j < ∞)  //traverse the bay
8          nextX = x−coord of intersection of L_i, L_j
9          if (nextX < currentX)
10             if (nextX is to the right of the cut)
11                 currentX = nextX
12                 memo = j
13             j = lht[j]
14         else
15             break            //break out the while loop
16 lht[i] = memo
```

Listing 2: Pseudo-code of updating the LHT

The time for processing one ready pair is $O(n)$. The total time for the complete sweep would be $O(n^3)$ but is only $O(n^2)$ via amortized analysis. Recall that the lines in the arrangement are sorted by increasing slopes. L_i will participate in $n-i$ pairs as the upper line and will perform traversal of the UHT. It will participate in $i-1$ pairs as the lower line and will perform traversal of the LHT. Consider a fixed L_i. Aggregating over all of the bay traversals of the UHT and LHT, the `while` loops will iterate at most $3(n-1)$ times in total. Thus, the time for processing the intersections on one line is $O(n)$, for a total time of $O(n^2)$ for the sweep.

3 Parallel Topological Sweep

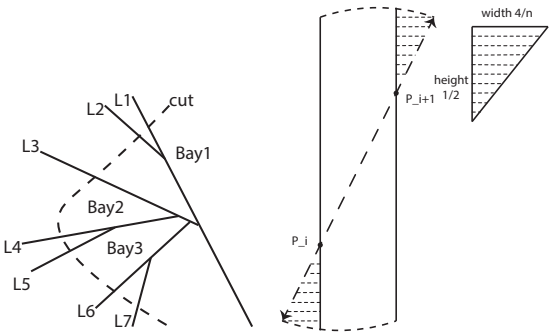
Multiple ready pairs can be processed in parallel. The task is to keep the horizon trees consistent. Fig. 3(a) is an LHT with three ready pairs: (L_1, L_2), (L_4, L_5), and (L_6, L_7). Recall that in updating the LHT, the

right endpoints of the upper lines — L_1, L_4, and L_6 — remain unchanged. The lower lines need to traverse the bays beyond their partners to find new endpoints. When L_2 crosses beyond L_1, it will proceed to infinity. When L_5 crosses beyond L_4, it will meet L_3. Even if L_3 is absent from the arrangement, L_5 can not intercept L_2 before L_2 intersects with L_1. Otherwise, L_2 would not be a ready partner with L_1. Similarly, when L_7 crosses beyond L_6, it can not intercept L_5 nor L_2. Thus, the lines coming from below — L_2, L_5, and L_7 — may simultaneously traverse their bays for new intersections. These simultaneous traversals entail concurrent read of `lht[]` by multiple processors, which is innocuous. What is critical is that after a processor has finished its traversal, it must hold off updating its local region of `lht[]` until all traversals are done. Otherwise, there will be a race condition, as illustrated in the following scenario. Assume that L_5 finishes its traversal of Bay_2 and updates `lht[5]` to L_3 right away. At that moment, if L_7 has already moved from L_5 to L_4, no harm is done. If, however, `lht[5]` is overwritten before L_7 finishes with L_5, then L_7 will follow the *new* `lht[5]` to L_3 rather than L_4. This would break the algorithm. Thus, the processors must synchronize before they write to `lht[]`. When they do write, they write to different parts of `lht[]` — there is no risk of concurrent write. Note that if (L_6, L_7) is *sequentially* processed after (L_4, L_5), then L_7 will visit only L_4 and L_3. With parallel processing, L_7 will visit L_5 in addition to L_4 and L_3. This is a source of parallel overhead.

```
1  parFor r from 1 to numReady private(memoL,memoU)
2  //ready pair (cut[ready[r]], cut[ready[r]+1])
3  memoL = new endpoint of L_{cut[ready[r]+1]} in LHT
4  memoU = new endpoint of L_{cut[ready[r]]} in UHT
5  synchronize
6  lht[ cut[ ready[r] + 1 ] ] = memoL
7  uht[ cut[ ready[r] ] ] = memoU
8  //update closer[ready[r]] & closer[ready[r]+1]
9  //swap cut[ready[r]] and cut[ready[r]+1]
10 synchronize
11 //find new ready pairs; update ready & numReady
```

Listing 3: Pseudo-code of parallel topological sweep

Let the variable `numReady` be the number of ready pairs stored in the array `ready[1..numReady]`. Listing 3 describes how to process all ready pairs in parallel. The `parFor` loop on line 1 is executed by `numReady` processors simultaneously. Each processor works on one pair. The clause `private()` on line 1 reserves two private variables for *every* processor that can be read and written without contention with other processors. The imperative `synchronize` on lines 5 and 10 stipulates that all processors must finish the proceeding steps before any of them proceed further. At line 11, each processor looks above and below to see if its two lines will be engaged in new ready pairs. Care must be taken that a new pair is identified exactly once. For example, assume two processors work on two *adjacent* pairs $(L_{\text{cut}[i]}, L_{\text{cut}[i+1]})$ and $(L_{\text{cut}[i+2]}, L_{\text{cut}[i+3]})$. If $L_{\text{cut}[i]}$ and



(a) Parallel processing of ready pairs.

(b) A pair of ready points in the dual arrangement.

Figure 3: Parallel processing of ready pairs.

$L_{\text{cut}[i+3]}$, which will have become $L_{\text{cut}[i+1]}$ and $L_{\text{cut}[i+2]}$, form a new pair, only one processor should add it to `ready[]`. After peeking at each other's data, the first processor leaves this task to the second processor. Furthermore, the new ready pairs must be collated and saved consecutively at the front of `ready[]`. Because there may be zero, one, or two new pairs per existing one, the processors do not know in advance where to write in the array `ready[]`. This can be solved with prefix sum. For example, let this sequence [1 0 1 0 1 0] be the numbers of new pairs found by six processors. The exclusive prefix sum of this sequence is [0 1 1 2 2 3], which can be computed in $\lceil \log_2 6 \rceil$ steps using the parallel prefix sum algorithm [6]. Adding one to the exclusive prefix sum, the sequence [1 2 2 3 3 4] is the locations in the array `ready[]` where the processors can write down their ready pairs in parallel.

The computation in Listing 3 constitutes one stage of parallel topological sweep that processes one batch of ready pairs and produces the next batch. The algorithm repeats stage after stage until all $\binom{n}{2}$ pairs are processed. Although parallelization incurs some overhead, the overall time remains $O(n^2)$ if the parallel computation is serialized. Parallelization does not change the observation that the `while` loops for bay traversals will iterate at most $3(n-1)$ times on behalf of each line.

4 Expected Concurrency

This section studies the expected number of ready pairs at the first stage. The lines are generated via the dual. The dual points are uniformly distributed in the interior of the unit circle, excluding the origin. For each point, its polar angle is a uniform random number between 0 and 2π. Its distance to the origin is the square root of a uniform random number in the open interval $(0, 1)$. Square root is taken because the area is proportional to the square of the distance. The polar coordinates are converted to the Cartesian coordinates, which become the coefficients of the lines in the primal arrangement.

This section, however, works with the dual.

Let $x_1, \ldots, x_n$ be the sorted x-coordinates of the points $P_1, \ldots, P_n$. The x_i's are distinct because identical ones would result in parallel lines in the primal arrangement, which is assumed to be simple. The n vertical lines, $x = x_i$, shred the unit circle into $n+1$ vertical strips. At least $\lceil n/2 \rceil$ strips have widths less than $4/n$. Otherwise, the strips would be wider than the unit circle. Fig. 3(b) shows such a strip. Recall that walking a line from left to right corresponds to rotating two hands around the dual point. The UHT corresponds to the upper hands that start at twelve o'clock. The LHT corresponds to the lower hands that start at six o'clock. They rotate clockwise and stop at other points. The set-intersection of the UHT and LHT is to take the smaller rotation of the two hands. For example, if the upper hand has rotated to two o'clock and the lower hand to seven o'clock, the smaller rotation is $\pi/6$. In Fig. 3(b), if the duals of P_i and P_{i+1} form a ready pair at the first stage, their four hands have the following configuration. First, the upper hand of P_i has rotated to P_{i+1}, and the antipodal image of its lower hand has rotated beyond P_{i+1}. Second, the lower hand of P_{i+1} has rotated to P_i, and the antipodal image of its upper hand has rotated beyond P_i. If the pair is not ready, at least one of their hands has stopped at a third point that resides in one of the shaded pies below P_i and above P_{i+1}.

Lemma 1 *Assume P_i and P_{i+1} are in the bottom and top quarters, respectively, of their vertical lines. If the strip between P_i and P_{i+1} has a width less than $4/n$, the area of the shaded pie above P_{i+1} is at most $1/n$. So is the area of the shaded pie below P_i.*

Proof. When P_{i+1} is in the top quarter of its vertical line, the length of the vertical side of the shaded pie above P_{i+1} is at most $1/2$. The horizontal span of the arc is at most the width of the strip, $4/n$. Thus, the pie fits inside the right-angled triangle with an area of $1/n$. So is the pie below P_i. □

Let X_i, $i = 1, \ldots, n-1$, be the indicator that the duals of P_i and P_{i+1} are a ready pair. Let X be the random variable of the number of ready pairs at the first stage.

Theorem 2 *If the n dual points are uniformly distributed in the unit circle, the expected number of ready pairs at the first stage is $\Omega(n)$.*

Proof. The probability P_i and P_{i+1} are in the bottom and top quarters, respectively, of their vertical lines is $1/4 \cdot 1/4 = 1/16$. If the strip has a width less than $4/n$, the probability a point is in a shaded pie below P_i or above P_{i+1} is less than $(2/n)/\pi = 2/(\pi n)$. The probability the duals of P_i and P_{i+1} are a ready pair is

$$\Pr(X_i = 1) \geq \frac{1}{16}\left(1 - \frac{2}{\pi n}\right)^{n-2}.$$

Although at least $\lceil n/2 \rceil$ strips have widths less than $4/n$, the leftmost and rightmost strips must be excluded. Both of them are demarcated by only one point.

$$\begin{aligned}
\mathrm{E}(X) &= \sum_{i=1}^{n-1} \mathrm{E}(X_i) \\
&\geq \left(\left\lceil \frac{n}{2} \right\rceil - 2\right) \frac{1}{16}\left(1 - \frac{2}{\pi n}\right)^{n-2} \\
&= \left(\left\lceil \frac{n}{2} \right\rceil - 2\right) \frac{1}{16} \exp\left((n-2) \cdot \ln\left(1 - \frac{2}{\pi n}\right)\right) \\
&\geq \left(\left\lceil \frac{n}{2} \right\rceil - 2\right) \frac{1}{16} \exp\left((n-2)\frac{\frac{-2}{\pi n}}{1 + \frac{-2}{\pi n}}\right) \\
&\geq \left(\frac{n}{2} - 2\right) \frac{1}{16} \exp\left(-\frac{2n-4}{\pi n - 2}\right) \\
&\geq \left(\frac{n}{2} - 2\right) \frac{1}{16e}
\end{aligned}$$

□

By the theorem, the expected number of ready pairs is at least $0.01n$. Empirical data suggest there are more than that. Fig. 4(a) shows the boxplots of the numbers of ready pairs at the first stage for n from 10,000 to 200,000. For each n, 100 random arrangements are generated, and their ready pairs at the first stage are counted. In the boxplots, the numbers of pairs are divided by n and become fractions. The central mark of a box indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The medians are at least $0.297n$ for all n.

The author does not know the expected number of ready pairs after the first stage. Fig. 4(b) shows the boxplots of the average numbers of ready pairs per stage for n from 20,000 to 400,000. For each n, ten random arrangements are generated. For each arrangement, the number $\binom{n}{2}$ is divided by the number of stages, resulting in the average number of ready pairs per stage, which is then divided by n and becomes a fraction. The medians are at least $0.153n$. If the expected number of ready pairs per stage is $\Omega(n)$, parallel topological sweep will run in $\mathrm{O}(n)$ time using $\lfloor n/2 \rfloor$ processors. Empirical data suggest that there are $3.3n$ stages on average.

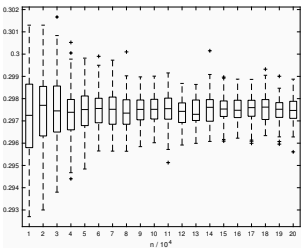
5 Implementations

There are three implementations in the literature. The Rosenberger code [3, 12] is difficult to locate. The Miller code [8] has a broken URL. The Rafalin code [11] is downloaded from the Tufts University website [10]. It is slightly revised and brought up to the latest language standard. Two new implementations are developed. A new serial code is implemented in C. The first parallel code is implemented in C and OpenMP. The two new implementations are available on GitHub,

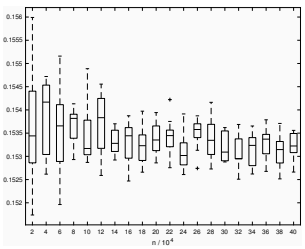
`github.com/mingouyang/parTopoSwp`. The code is compiled with the Intel C compiler (`icc` and `icpc 19.0.5.281`) with the optimization flags `-O3 -xHost -ipo`. The computation is performed on a CentOS 7 server with two Intel Xeon 6150 Skylake 2.70 GHz 18-core CPUs and 384GB DDR4 2,666 MHz RAM.

Fig. 4(c) compares the performance of the three implementations. For each implementation at each value of n, the average runtime of ten random arrangements is plotted. The Rafalin code solves $n = 100,000$ in 692.2 second. The new serial code solves $n = 200,000$ in 742.8 second. On average, the new serial code is 3.92 times faster than the Rafalin code. The parallel code is executed with 64 OpenMP threads. It solves $n = 400,000$ in 678.3 second. It is more than 16 times faster than the Rafalin code for large n.

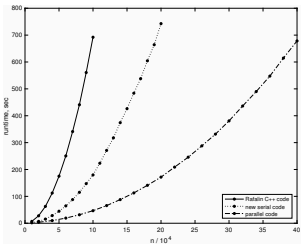
Fig. 4(d) shows the speedup curve of the parallel code using 2, 4, 8, 16, 32, and 64 threads when n is fixed at 300,000. The baseline is the new serial code — it uses one thread. Speedup is calculated as the serial runtime divided by the parallel runtime. When the number of threads is two and four, the parallel code runs *slower* than the serial code. This comes from the parallel overhead described in Section 3 as well as the penalty of thread synchronization. At each synchronization point, some threads will be waiting for the others to finish their work. Their idling is another form of parallel overhead. With 64 threads, the parallel code is 4.2 times faster than the serial code. The server has 36 physical cores. The hyper-threading technology of Intel allows two threads to share a physical core. Thus, the hardware may support up to 72 threads. When sharing cores, however, the threads rarely run as fast as when each thread occupies a physical core exclusively. It is likely that if there are 64 physical cores, the parallel code may reach higher performance.



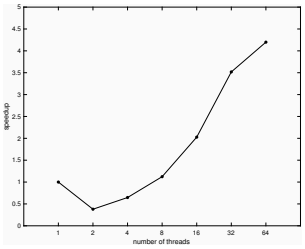
(a) For n from 10,000 to 200,000, the numbers of ready pairs at the first stage divided by n. Each box has 100 random arrangements.



(b) For n from 20,000 to 400,000, the average numbers of ready pairs per stage divided by n. Each box has ten random arrangements.



(c) The Rafalin code is the slowest. The parallel code is the fastest. The new serial code is in between.



(d) The speedup curve of the parallel code for numbers of threads from 2 to 64 with n fixed at 300,000.

Figure 4: Empirical results.

6 Discussion

Topological sweep [3] is a building block of some efficient algorithms. The present work is the first to parallelize it. The classical algorithm processes the ready pairs one at a time. Herein it is shown that ready pairs can be processed in parallel. For random arrangements, it is proved that the expected number of ready pairs at the beginning is $\Omega(n)$. Empirical data suggest that the average number of ready pairs for the rest of the parallel stages is also $\Omega(n)$. If this is proved, topological sweep can be done in expected linear time using $\lfloor n/2 \rfloor$ processors. Arrangements that constrict concurrency can be constructed. The number of parallel stages is the maximal monotone path length, which is $\Omega(n^{2-o(1)})$ [1, 2].

Three implementations are compared. The code by Rafalin *et al.* [11] is designed to handle degenerate arrangements. A new serial code for simple arrangements is implemented that runs as fast as the author can make it. It will underreport the vertices of degenerate arrangements. When compiled with the same compiler and executed on the same hardware, the new serial code is 3.92 times the speed of the Rafalin code. The speedup is attained by ignoring degeneracy as well as by streamlined computation. Based on the new serial code, the first parallel code is implemented in C and OpenMP. When executed with 64 threads, it is 4.2 times the speed of the serial code and more than 16 times that of the Rafalin code. Both new implementations are available on GitHub. The parallel code is merely a proof of concept. For future work, the code can be sped up with advanced techniques of parallel programming, such as dynamic load balancing and non-uniform memory access tuning.

Graphics processing units have thousands of vector processors. They are designed for massive single-instruction-multiple-data (SIMD) computation. The parallel traversals of the horizon trees loosely fit the SIMD paradigm. The complication is that the bays to be explored have different sizes. Some processors may finish their work before the others. This divergence in computation is a source of SIMD overhead. It is an interesting problem to reorganize the traversals so that such overhead is reduced.

References

[1] Balogh, J., Regev, O., Smyth, C., Steiger, W., Szegedy, M.: Long monotone paths in line arrangements. Discrete & Computational Geometry. **32**(2), 167–76 (2004).

[2] Dumitrescu A.: On some monotone path problems in line arrangements. Computational Geometry **32**(1), 13–25 (2005).

[3] Edelsbrunner, H., Guibas, L.J.: Topologically sweeping an arrangement. Journal of Computer and System Sciences **38**(1), 165–194 (1989).

[4] Edelsbrunner, H., Souvaine, D.L.: Computing least median of squares regression lines and guided topological sweep. JASA **85**(409), 115–119 (1990).

[5] Gil, J., Steiger, W., Wigderson, A.: Geometric medians. Discrete Mathematics **108**(1-3), 37–51 (1992).

[6] Hillis, W.D., Steele Jr, G.L.: Data parallel algorithms. Communications of the ACM **29**(12), 1170–1183 (1986).

[7] Liu, R.: On a notion of data depth based on random simplices. The Annals of Statistics **18**(1), 405–414 (1990).

[8] Miller, K., Ramaswami, S., Rousseeuw, P., Sellarès, J.A., Souvaine, D., Streinu, I., Struyf, A.: Efficient computation of location depth contours by methods of computational geometry. Statistics and Computing **13**(2), 153–162 (2003).

[9] Pocchiola, M., Vegter, G.: Topologically sweeping visibility complexes via pseudotriangulations. Discrete & Computational Geometry **16**(4), 419–453 (1996).

[10] Rafalin, E.: Topological sweep in degenerate cases. `www.cs.tufts.edu/r/geometry/other/sweep/` (2002), [Online; accessed 2020-1-9].

[11] Rafalin, E., Souvaine, D., Streinu, I.: Topological sweep in degenerate cases. In: Workshop on Algorithm Engineering and Experimentation, pp. 155–165, Springer (2002).

[12] Rosenberger, H.: Topological plane sweep implemented in C. University of Illinois at Urbana-Champaign (1990).

One Hop Greedy Permutations*

Donald R. Sheehy†

Abstract

We adapt and generalize a heuristic for k-center clustering to the permutation case, where every prefix of the ordering is a guaranteed approximate solution. The *one-hop greedy permutations* work by choosing at each step the farthest unchosen point and then looking in its local neighborhood for a point that covers the most points at a certain scale. This balances the competing demands of reducing the coverage radius and also covering as many points as possible. This idea first appeared in the work of Garcia-Diaz et al. [6] and their algorithm required $O(n^2 \log n)$ time for a fixed k (i.e. not the whole permutation). We show how to use geometric data structures to approximate the entire permutation in $O(n \log \Delta)$ time for metrics sets with spread Δ. Notably, this running time is asymptotically the same as the running time for computing the ordinary greedy permutation.

1 Introduction

Greedy permutations of points in a metric space are useful for many standard computations, such as proximity search data structures, sampling, and k-center clustering. They were developed independently by Gonzalez [7] and also Dyer and Frieze [4] in 1985 for k-center clustering. Starting from any point the next point in a greedy permutation is chosen to be the farthest remaining point from the previously chosen points. Greedy permutations are effective for so many tasks because every prefix of the ordering gives a good, mostly uniform sample, keeping points as far apart as possible while (approximately) minimizing the maximum distance from any point to the sample.

The coverage radius of a subset $S \subset P$ is the minimum r such that $P \subseteq \bigcup_{x \in S} \texttt{ball}(x, r)$. The metric k-center problem is a search for k points that minimize the coverage radius. The first k points of a greedy permutation provide a 2-approximate k-center, i.e. the coverage radius is at most twice the optimal solution. This is known to be the best possible unless $P = NP$. However, there are several heuristics that have been shown to produce better solutions in practice. One such heuristic developed by Garcia-Diaz et al.[6] achieves a worse theoretical guarantee, but consistently outperforms the greedy approach on benchmarks. That approach works for a fixed r by choosing at each step, not the farthest point, but instead a point within distance r of the farthest point that covers the most previously uncovered points in its radius r ball. Then the algorithm binary searches for a good value of r. We call this the *one-hop k-center* algorithm.

Given that the greedy permutation is widely used because of its ability to provide a sequence of good covers, it makes sense to import these efficient heuristics from the setting with fixed r and k into the permutation setting. In this paper, we will generalize this approach to give a permutation of one-hop k-centers and show how to relate it to approximate greedy permutations. We call these *one-hop greedy permutations.* We will then show how to compute a one-hop greedy permutation in $O(n \log \Delta)$ time, where Δ is the *spread* of the input (the ratio of the largest to smallest pairwise distances). As (almost) always with such an analysis, the true worst case is $O(n^2)$, but would require point sets with exponential spread to achieve. In theory, this can be brought down to $O(n \log n)$ using elaborate theoretical techniques [8]; however, given that our interest is in the practical performance, we describe our algorithm in terms of a standard practical approach. In theory, the greedy approach is optimal anyways. In practice, it is very rare to find inputs with super-polynomial spread. A proof of concept implementation was used to generate some examples visualized in Section 5.

2 Background

2.1 Metrics Spaces

Let P be a finite subset of a metric space. Let $\mathbf{d}(a, b)$ denote the distance between a and b. The *metric ball* centered at x with radius r is

$$\texttt{ball}(x, r) := \{p \in P \mid \mathbf{d}(x, p) \leq r\}.$$

For a subset $S \subset P$, let $\mathbf{d}(a, S) := \min_{x \in S} \mathbf{d}(a, x)$. The *Hausdorff* distance between two subsets is defined as

$$\mathbf{d}_H(S, T) := \max\{\max_{s \in S} \mathbf{d}(s, T), \max_{t \in T} \mathbf{d}(t, S)\}.$$

So, for a subset $S \subset P$, this simplifies to $\max_{p \in P} \mathbf{d}(p, S)$, also known as the *coverage radius* of S. A subset $S \subseteq P$ is an (α, β)-*net* if it has coverage

*This work was partially supported under grant CCF-1652218.

†Department of Computer Science, North Carolina State University, don.r.sheehy@gmail.com

radius at most β and all pairs of points are at least α apart. The constant α controls the *packing* and β controls the *covering*. If $\alpha = \beta$, then we call it an α-net, and if the constants are unimportant, we just call it a net. The *spread* Δ of a point set is the ratio of the largest distance to the smallest distance. The quantity $\log \Delta$ features naturally in the analysis of many geometric algorithms and data structures as it roughly counts the number of levels in a hierarchical data structure in which the scale drops by a constant factor at each level.

The *doubling dimension* for a metric is the minimum number ρ such that every ball of radius $2r$ can be covered by 2^ρ balls of radius r. A metric is said to be a *doubling metric* if the doubling dimension is bounded by a constant. The natural appeal of doubling metrics is that they give a notion of low dimensionality for general metric spaces. In particular, packing and covering arguments similar to those used in Euclidean space can be used. For example, an (α, β)-net of the points in a ball of radius r will have at most $O(2^{r/\alpha})$ points.

2.2 From Approximate Greedy to k-Center

Let $P = (p_1, \ldots, p_n)$ be an ordered set of points and let $P_i = (p_1, \ldots, p_i)$ be the ith *prefix*. A *c-approximate greedy permutation* is an ordering of P such that for all $i = 1 \ldots n-1$,

$$\mathbf{d}_H(P_i, P) \leq c\mathbf{d}(p_{i+1}, P_i).$$

A 1-approximate greedy permutation is simply called a *greedy permutation*.

Below, we explain how the standard proof that the greedy permutation gives 2-approximate k-centers for all k can be extended to the approximate greedy case. More specifically, we show that that a c-approximate greedy permutation yields a $2c$-approximate k-center clustering.

Lemma 1 *If P is ordered according to a c-approximate greedy permutation, then every prefix P_k is an $(\frac{r}{c}, r)$-net where $r = \mathbf{d}_H(P_k, P)$.*

Proof. Fix any value of k. The coverage radius of P_k is r by definition. For the packing condition, we observe that for any $i < j \leq k$, we have

$$\begin{aligned}\mathbf{d}(p_i, p_j) &\geq \mathbf{d}(p_j, P_{j-1}) \\ &\geq \frac{1}{c}\mathbf{d}_H(P, P_{j-1}) \\ &\geq \frac{1}{c}\mathbf{d}_H(P, P_k) \\ &= \frac{r}{c}.\end{aligned}$$

□

Lemma 2 *If $P = (p_1, \ldots, p_n)$ be a c-approximate greedy permutation for some $c \geq 1$, then P_k is a $2c$-approximate k-center for all k.*

Proof. Let $r = \mathbf{d}_H(P_k, P)$ be the coverage radius of P_k. Let r_* be the radius of the optimal k-center, Opt. The goal is to show that $r \leq 2cr_*$.

By Lemma 1, every two points in P_k are at least $\frac{r}{c}$ apart. If there are two points of P_k within distance r_* of one center in the optimal solution, then their distance is at most $2r_*$. Thus, it would follow that $\frac{r}{c} \leq 2r_*$ and therefore $r \leq 2cr_*$ as desired. If no two points have this property, then every ball of radius r_* in the optimal solution contains a unique point of P_k and thus $\mathbf{d}_H(P_k, Opt) \leq r_*$. By the triangle inequality, $r \leq 2r_* \leq 2cr_*$ because

$$r = \mathbf{d}_H(P, P_k) \leq \mathbf{d}_H(P_k, Opt) + \mathbf{d}(P, Opt) \leq 2r_*.$$

□

3 The One-Hop Greedy Permutation

The heuristic proposed by Garcia-Diaz et al., which we refer to as *one-hop k-center* was motivated by the observation that the greedy permutation achieves its factor of 2 approximation factor by choosing points very conservatively. Another perspective is that the greedy approach reduces the coverage radius by actively seeking out and covering the extremes, which is at odds with the goal of finding "centers".

A simple one-dimensional example shows how the greedy algorithm can make poor selections (see Figure 1). Suppose we start with a unit-length line segment densely sampled with points and a sample point on one end. The next point taken in the greedy ordering is the other end of the segment. The coverage radius is one half. A better choice would be to take the point at $\frac{2}{3}$. This would reduce the coverage radius to $\frac{1}{3}$.

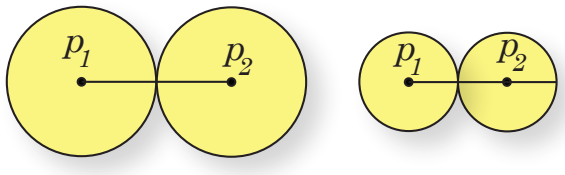


Figure 1: On the left, a greedy permutation takes the end point as its second point, resulting in a coverage radius of $\frac{1}{2}$. On the right, the one-hop greedy permutation backs of from the purely greedy choice resulting in a coverage radius of $\frac{1}{3}$.

There are several challenges that naturally arise in generalizing the one-hop approach from fixed radius (i.e. fixed k) setting to the permutation setting. The main

one is the changing radius; the radius decreases with each prefix. This means we must choose what the target radius should be at each step rather than using binary search as in the original work. If we did use binary search, we would potentially get a completely different set for each k instead of a single permutation of the input. The natural choice for the scale at each step is to be some fraction of the current radius. We will allow this to be a tunable parameter, which also allows us to prove some other theoretical properties below.

Let P be the input metric. We will define an ordering $(p_1, \ldots p_n)$ of P. Let $P_i = (p_1, \ldots, p_i)$ denote the prefixes of the ordering. For each i, let q_i denote the point in P that maximizes the distance to P_{i-1}. In other words, q_i is the point that would be added next if the permutation were greedy. Fix a parameter α. The ordering is *α-one-hop greedy* if for each i, the point p_i is the point in $\texttt{ball}(q_i, \alpha r)$ with $r = \mathbf{d}(q_i, P_{i-1})$ that maximizes

$$\left| \texttt{ball}(p_i, \alpha r) \setminus \bigcup_{j=1}^{i-1} \texttt{ball}(p_j, \alpha r) \right|.$$

The point is to balance between covering the farthest point and also covering many points.

Lemma 3 *If P is ordered according to an α-one-hop greedy permutation, then P is $\frac{1}{1-\alpha}$-approximate greedy.*

Proof. By definition, the point p_i is chosen in a ball of radius αr centered at the point of distance $r = \mathbf{d}_H(P_{i-1}, P)$ from P_{i-1}. So, by the triangle inequality, $\mathbf{d}(p_i, P_{i-1}) \leq (1-\alpha)r = (1-\alpha)\mathbf{d}_H(P_{i-1}, P)$. Therefore, P is $\frac{1}{1-\alpha}$-approximate greedy as desired. □

The Lemma 2 and Lemma 3 imply that our one-hop greedy permutations will give $\frac{2}{1-\alpha}$-approximate k-center solutions. As α goes to zero, we get the standard greedy permutation and its corresponding 2-approximate k-centers.

The choice of α is nonobvious. On the one hand, a large value provides flexibility in choosing the next point. On the other hand, that takes more time and the guarantee gets worse. Algorithmically, the cleanest choice for α is to let it equal $\frac{1}{3}$. For any value of $\alpha \leq \frac{1}{3}$, the points in $\texttt{ball}(p_i, \alpha r)$ are disjoint from $\bigcup_{j=1}^{i-1} \texttt{ball}(p_j, \alpha r)$. This saves us the trouble checking if points are already covered when choosing the next point. Perhaps accidentally, choosing $\alpha = \frac{1}{3}$ leads to a 3-approximate k-center, which is the approximation ratio achieved by Garcia-Diaz et al. in the fixed radius case. Whether such a choice is actually best will likely vary based on peculiarities of the input.

4 Efficient Approximations

The original algorithm for one-hop k-centers started by sorting all distances in $O(n^2 \log n)$ time. It then, used this ordering to compute a one-hop k-center for a given radius r in quadratic time in a manner similar to the original Gonzalez greedy ordering algorithm. The extra step in each iteration was the search in the neighborhood of the greedy choice for a point that covers more points in its radius r ball (that were not already covered by a previously inserted point). This also takes quadratic time. Thus, the total running time is $O(n^2 \log n)$.

In the low-dimensional setting, one can hope to do better with approximations. In the case of greedy permutations, Har-Peled and Mendel [8] showed that the approach of Clarkson [2, 3] runs in $O(n \log \Delta)$ time. We will augment this approach with a data structure that does approximate range sampling, to make the local improvement step faster. In the end, we will have an algorithm that runs in $O(n \log \Delta)$ time. Thus, it matches the asymptotic running time of the greedy permutation computation, albeit with worse constants.

A $(1+\varepsilon)$-*approximate one-hop greedy permutation* of P is an ordering of P that could be a one-hop greedy permutation if all distances are perturbed by a factor of at most $1+\varepsilon$. For such an approximation, it will suffice to choose the next point among an εr-net where r is the distance to the current farthest point. It also suffices to count points in metric balls only approximately as described below.

4.1 Approximate Range Sampling

A *metric range search* takes a point x and a radius r as input and returns the points in $\texttt{ball}(x, r)$. A *c-approximate metric range search* returns a set of points S such that

$$\texttt{ball}(x, r) \subseteq S \subseteq \mathit{ball}(x, cr).$$

That is, it may return some extra points that are close to the desired ball. Similarly, *metric range counting* and *c-approximate range counting* return the number of points that would be returned by the corresponding search.

Many data structures for proximity search on metric spaces can easily be adapted to perform *approximate range sampling*, which is a hybrid between range search and range counting. A range sampling query (x, r, ε) for a given resolution ε returns an ε-net of $\texttt{ball}(x, r)$ along with a weight for each point. The points in the range are each assigned to a point in the sample within distance ε and the weight of a point is the number of points assigned to it. In the c-approximate variant, the range may include points of distance up to cr away from x.

Perhaps the simplest data structure to use for approximate range sampling would be the cover tree of Beygelzimer et al. [1], particularly in the variant by Izbicki and Shelton [9]. Other hierarchical data structures such as navigating nets [12], net-trees [8, 11, 10], or deformable spanners [5] could also be used. In any of these data structures, one searches through a hierarchy of nets, where the ith level is a 2^i net. They are all designed for range searching, though they are sold as nearest neighbor search data structures, which is, of course, a kind of range search with a range that shrinks as you proceed. A search through such a data structure maintains a 2^i-approximate range sample at each level. Moreover, for doubling metrics, the number of points in the sample is $2^{O(2^i/r)}$. So, any search that stops at a level i such that $r = \Omega(2^i)$ will only return a constant sized set of (weighted) points. The running time of such a search is proportional to the number of levels searched, which is $O(\log \Delta)$ in the worst case.

4.2 Putting it all together

A standard approach to computing a greedy permutation is to use a kind of discrete Voronoi diagram. At step i, each point is associated with its nearest neighbor in P_i. A *cluster* is the set of points associated with a given point. When the point p_i is added to the permutation, a search is performed to find which points now have p_i as their nearest neighbor. To speed up the search, one stores a graph with vertex set P_i that connects two points p_a and p_b if adding a point from the cluster of p_a could affect the cluster of p_b. Armed with this graph, the update only checks points within a constant factor of the current coverage radius. This approach first formalized [2] and implemented [3] by Clarkson was shown to only require $O(n \log \Delta)$ time for greedy permutations in doubling metrics by Har-Peled and Mendel [8]. The same analysis easily holds for approximately greedy permutations. The two key steps are that, one, the graph has constant degree on a net, and, two, the distance to a point is computed only a constant number of times before the coverage radius must go down by a factor of two. We call this structure the *cluster graph*.

Our algorithm uses a cluster graph and approximate range sampling to compute an approximate one-hop greedy permutation. Let $\alpha < 1$ be the hop parameter. Let $\varepsilon <= 1$ be the desired approximation factor. At each iteration, we add one point using the following steps. First, we use a heap to quickly find the existing point p whose cluster has the largest radius. The farthest point f in this cluster is the point that would be added in a pure greedy permutation. Let r be the distance from f to p. We compute an approximate range sample S in $\texttt{ball}(f, 2\alpha r)$ at scale εr. We can then compute approximate range counts for $\texttt{ball}(q, \alpha r)$ for each $q \in S$ in constant time by adding the weights of points in S, careful not to count points that are within αr of a previously added point. The previously added points that could be within this radius are all neighbors of p in the cluster graph, so there are only a constant number of them. There are a constant number of points in S and each takes constant time. We add the point with the largest count. In total, we get the following.

Theorem 4 *Let P be points in a metric space with constant doubling dimension. Let $\alpha < 1$ and $\varepsilon < 1$ be constants. Then, a $(1+\varepsilon)$-approximate α-one-hop greedy permutation of P can be computed in $O(n \log \Delta)$ time.*

5 A Proof of Concept

We integrated an implementation of the one-hop greedy permutations into our Python library for greedy permutations. It can be found on Github at https://github.com/donsheehy/greedypermutation. This library also includes standard algorithms for computing pure greedy permutations and its approximations.

We found that on small, low-dimensional examples, there appears to be a very slight improvement with the one-hop greedy permutations. The example in Figure 2 is typical. We set $\alpha = 1/3$ and took a uniform sample of points in a square. The figure show three different scales in the permutation at $k = 10$, $k = 25$, and $k = 50$. The graph in Figure 3 shows how the radii change with the number of points. Even in this simple example, the one-hop greedy permutation often has the smaller coverage radius.

It remains to see if the improvements attained with one-hop k-centers on large benchmark instances coming from TSP datasets [6] are realized by the one-hop greedy permutations.

6 Conclusions

We have generalized the one-hop k-center heuristic to define one-hop greedy permutations. We have also given an efficient algorithm to compute approximations in $O(n \log \Delta)$ time.

There are several future directions to consider. In our implementation, we used a standard greedy permutation to build the data structure for approximate range sampling. This is novel in that it does a kind of boot strapping from an ordinary greedy permutation to a one-hop greedy permutation. Another direction to consider is whether the final choice should really count points by weight or just count points in the approximate range sample. It seems that the latter approach, though farther from the one-hop k-center approach could be more stable to drastically varying density in the underlying points set.

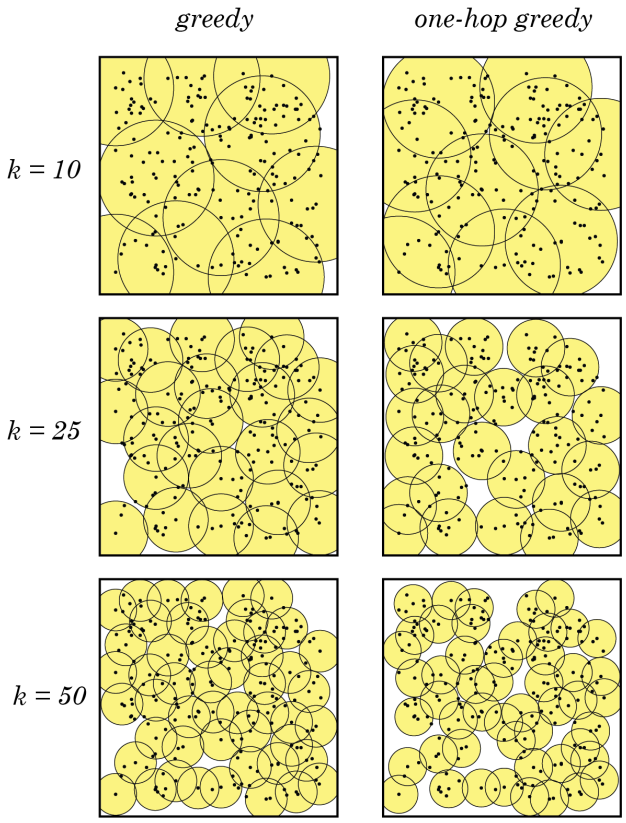


Figure 2: For points in the plane, there is little visual difference between the two cases except at the boundary. The one-hop greedy permutation tends to overshoot the boundary less.

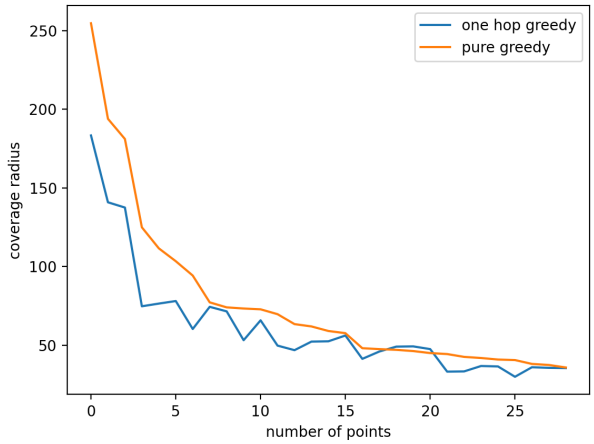


Figure 3: This is the graph of the coverage radii for the two approaches on the small example shown above.

References

[1] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.

[2] K. L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

[3] K. L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for 'sb(s)'. Preliminary version presented at ALENEX99, 2003.

[4] M. Dyer and A. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985.

[5] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Applications*, 35:2–19, 2006.

[6] J. Garcia-Diaz, J. Sanchez-Hernandez, R. Menchaca-Mendez, and R. Menchaca-Mendez. When a worse approximation factor gives better performance: a 3-approximation algorithm for the vertex k-center problem. *Journal of Heuristics*, 23(5):349–366, 2017.

[7] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[8] 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.

[9] M. Izbicki and C. R. Shelton. Faster cover trees. In *Proceedings of the Thirty-Second International Conference on Machine Learning*, 2015.

[10] M. Jahanseir and D. Sheehy. Nettrees. Available from http://dx.doi.org/10.5281/zenodo.1409233, 2018.

[11] M. Jahanseir and D. R. Sheehy. Transforming hierarchical trees on metric spaces. In *Proceedings of the Canadian Conference on Computational Geometry*, 2016.

[12] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *SODA*, 2004.

A Degree 3 Plane 5.19-Spanner for Points in Convex Position

Davood Bakhshesh* Mohammad Farshi†

Abstract

Let S be a set of n points in the plane that is in convex position. In this paper, using the well-known path-greedy spanner algorithm, we present an algorithm that constructs a plane $\frac{3+4\pi}{3}$-spanner G of degree 3 on the point set S. Recently, Biniaz et al. (*Towards plane spanners of degree 3, Journal of Computational Geometry, 8 (1), 2017*) have proposed an algorithm that constructs a degree 3 plane $\frac{3+4\pi}{3}$-spanner G' for S. We show that there is no upper bound for the total weight of G', but the total weight of G is asymptotically equal to the total weight of the minimum spanning tree of S.

1 Introduction

Let S be a set of points in the plane. A weighted graph G with vertex set S is called *geometric*, if any edge (p, q) of G is the straight line between p and q, and its weight is $|pq|$, the Euclidean distance between p and q. The total weights of the graph G is the sum of the weight of all edges of G and is denoted by $wt(G)$. Let $t > 1$ be a real number. The geometric graph G is called *t-spanner* for S, if for any two vertices p and q in G, there exists a path P between p and q in G such that $|P| \leq t|pq|$, where $|P|$ denotes the length of the path P which is the sum of the weight of all edges on P. For any two points u and v in a geometric graph G, let $\delta_G(u, v)$ be the length of the shortest path between u and v in G. The *stretch factor* (*dilation*) between u and v is defined the ratio $\frac{\delta_G(u,v)}{|uv|}$ and we denote it by $SF_G(u, v)$. The stretch factor $SF(G)$ of a graph G is defined as

$$SF(G) = \max_{u,v \in G} SF_G(u, v).$$

Note that when a geometric graph G is t-spanner, clearly $SF(G) \leq t$. We refer the reader to the book [13] for an overview of t-spanners and the related algorithms.

A *plane* spanner of bounded *degree* is a spanner whose edges do not cross each other and whose maximum degree is bounded by a constant. In Table 1, some of the results related to the plane spanner of bounded degree are summarized. Note that since the stretch factor of a Hamiltonian path through a set of points arranged in a grid is $\Omega(\sqrt{n})$ (see [13]), the lower bound on the maximum degree of a t-spanner is 3. The lower bound of the maximum degree of a t-spanners for points in convex position is also 3 (see [11]). Das and Heffernan [9] proved that spanners of maximum degree 3 always exist.

Points in Non-Convex or Convex Position		
Reference	Degree	Upper bound on the stretch factor
Bose et al. [7]	27	≈ 8.27
Li and Wang [12]	23	≈ 6.43
Bose et al. [8]	17	≈ 23.56
Kanj and Perkovic [14]	14	≈ 2.91
Bonichon et al. [3]	6	6
Bose et al. [5]	6	≈ 81.66
Bonichon et al. [4]	4	≈ 156.82
Kanj et al. [11]	4	20
Points in Convex Position		
Kanj et al. [11]	3	20
Biniaz et al.[1]	3	≈ 5.19

Table 1: Results of bounded degree plane spanners.

One of the famous algorithms for constructing a t-spanner on a given point set S is the *path-greedy spanner* algorithm or *greedy spanner* algorithm for short. The algorithm is as follows. First, the algorithm sorts all pairs of points in nondecreasing order of their Euclidean distance. Assume that the sorted data is stored in a list L. Let E be the edge set of the graph computed by the algorithm. First, the edge set E is considered empty. Next, the algorithm processes the pairs of points in L in order. Suppose that the algorithm wants to process the pair $(p, q) \in L$. If the length of the shortest path between p and q in the graph computed so far is greater than $t|pq|$, the algorithm adds the pair (p, q) to E; otherwise, the algorithm processes the next pair of points in L. The computed graph by the algorithm is called the *path-greedy* spanner or the *greedy spanner*. Algorithm 1, PathGreedy(S, t), shows the pseudocode of the greedy spanner algorithm (see Appendix). Now, we describe a modified version of the greedy spanner algorithm that we need later. In PathGreedy(S, t), the algorithm starts with a sorted list L and empty edge set E. The modified version of this algorithm, ModifiedPathGreedy(S, E, L, t)

*Department of Computer Science, University of Bojnord, Bojnord, Iran, dbakhshesh@gmail.com

†Combinatorial and Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran, mfarshi@yazd.ac.ir

Algorithm 3: Deg3PlaneSpanner(S) ([1])

input: A non-empty finite set S of points in the plane that is in convex position
output: A plane degree-3 spanner of S.

1 (p, q) := a farthest pair of points of S;
2 C_1, C_2 := the two chains obtained by removing p and q from $CH(S)$;
3 $E' := CH(S) \cup$ Matching(C_1, C_2);
4 **return** $G' = (S, E')$;

(see Appendix), takes two extra parameters: an edge set E and a sorted list L. If $E = \emptyset$ and L is the sorted list of all $\binom{n}{2}$ pairs of points of S in non-decreasing order of their distances, then the output of the two algorithms PathGreedy(S, t) and ModifiedPathGreedy(S, E, L, t) is same.

In 2017, Biniaz et al. [1] presented an algorithm that constructs a plane $\frac{3+4\pi}{3}$-spanner of maximum degree at most 3 for any set S of points in the plane that is in convex position. Their algorithm works as follows. At first, the convex hull of S, $CH(S)$, is added to the spanner. Then, it selects a farthest pair (p, q) of points of S. Then, it adds a special matching between two convex chains obtained by removing p and q from the convex hull (see Algorithm 3). The matching for the two convex chains is computed as follows. First, we compute the closest pair between two convex chains that are separated by a line. Given this closest pair, we split the two chains, and recurse on both sides.

In this paper, we focus on constructing a bounded-degree plane spanner for points in convex position of degree at most 3. Using the algorithm ModifiedPathGreedy, we propose an algorithm that constructs a plane $\frac{3+4\pi}{3}$-spanner of degree at most 3 for points in convex position. Then, we show that the proposed plane spanner can be computed in $O(n^2 \log n)$ time. In [1], Biniaz et al. did not mention the time complexity of their algorithm (Algorithm 3). In [2], Biniaz et al. present an $O(n \log^7 n)$-time algorithm that computes the plane spanner generated by Algorithm 3. We show that there is no upper bound on the total weight of the spanner proposed by Biniaz et al. [1], but using the concept of *generalized leapfrog property* (see [13]), we show that for any set S of points in the plane that is in convex position, the total weight of our proposed plane spanner is asymptotically equal to $wt(MST(S))$.

2 Preliminaries

In this section, we present some definitions and notations that we use in the following sections. Throughout the paper, we assume that S is a set of n points in the plane that is in convex position. A farthest pair (p, q) of points of S is called a *diametral pair*, p and q are called *diametral points* and the Euclidean distance $|pq|$ is called the *diameter* of S. We assume, without loss of generality, that the diametral pair (p, q) of S is horizontal and p is to the left of q. We denote the set of all points of $S \backslash \{p, q\}$, which are above the line segment pq and below pq by **upper** and **lower**, respectively. Let D_p and D_q be two closed disks with radius $|pq|$ centered at p and q, respectively. The intersection of D_p and D_q is denoted by $L(p, q)$ and is called the lune of p and q. In the following sections, we use the notation G to refer to the plane spanner proposed in the current paper, and G' to refer to the plane spanner generated by Algorithm 3. In the graphs G or G', an edge (a, b) is called a *shortcut edge*, if $a \in$ **upper** and $b \in$ **lower** or $a \in$ **lower** and $b \in$ **upper**.

3 A Degree 3 Plane Spanner for Points in Convex Position

In this section, we propose an algorithm that constructs a plane $\frac{3+4\pi}{3}$-spanner $G = (S, E)$ of degree at most 3 for S. The idea of the algorithm is as follows. The algorithm starts with $E = CH(S)$. Then, we run ModifiedPathGreedy(S, E, L, t), where $t = \frac{3+4\pi}{3}$ and L contains all pairs of points (a, b) with $a \in$ **upper** and $b \in$ **lower** and sorted by increasing the Euclidean distance $|ab|$ (see Figure 1). The graph G consists of $CH(S)$ and the output of ModifiedPathGreedy(S, E, L, t) (see Algorithm 4).

Algorithm 4: GreedyPlaneSpanner(S)

input: a set S of n points in the plane that is in convex position.
output: a plane $\frac{3+4\pi}{3}$-spanner G of degree at most 3.

1 (p, q) := a diametral pair of points of S;
2 **upper** := set of all points of S which are above the line segment pq;
3 **lower** := set of all points of S which are below the line segment pq;
4 L := list of all pairs of points (a, b) with $a \in$ **upper** and $b \in$ **lower** and sorted by increasing the Euclidean distance $|ab|$;
5 $E := CH(S) \cup$ ModifiedPathGreedy$(S, E, L, \frac{3+4\pi}{3})$;
6 **return** $G = (S, E)$;

Now, we prove that the output of algorithm GreeyPlaneSpanner(S) is $\frac{3+4\pi}{3}$-spanner. We start with the following lemmas, which are needed later.

Lemma 1 *Let S be a finite set of at least two points in the plane, and let (p, q) be any diametral pair of S. Then, the points of S lie in $L(p, q)$.*

Lemma 2 [1] *Let C be a convex chain with endpoints p and q. If C is in $L(p,q)$, then the stretch factor of C is at most $\frac{2\pi}{3}$.*

Theorem 3 *The graph G generated by* GreedyPlaneSpanner*(S) is a $\frac{3+4\pi}{3}$-spanner for S.*

Proof. Let $t = \frac{3+4\pi}{3}$ and let a and b be two arbitrary distinct points in S. To prove the theorem, it is sufficient to prove that $SF_G(a,b) \leq t$. Let (p,q) be the diametral pair of points selected in Line 1 of Algorithm 4. We consider two cases:

- **Case 1.** $a, b \in \textbf{upper} \cup \{p,q\}$ or $a, b \in \textbf{lower} \cup \{p,q\}$. We prove this case for $a, b \in \textbf{upper} \cup \{p,q\}$ (the other case is symmetric). Let C_u be the convex chain connecting p to q obtained by removing all points **lower** from $CH(S)$. By Lemma 1, C_u lies in $L(p,q)$. Then, by Lemma 2, $SF_{C_u}(a,b) \leq \frac{2\pi}{3}$. Since G contains $CH(S)$, $SF_G(a,b) \leq SF_{C_u}(a,b)$. Hence, $SF_G(a,b) \leq \frac{2\pi}{3} \leq t$.

- **Case 2.** $a \in \textbf{upper}$ and $b \in \textbf{lower}$, or $a \in \textbf{lower}$ and $b \in \textbf{upper}$. Suppose, without loss of generality, that $a \in \textbf{upper}$ and $b \in \textbf{lower}$. If (a,b) is an edge of G, then clearly $SF_G(a,b) = 1 \leq t$. Now, suppose that (a,b) is not an edge of G. Then, according to the construction of G, there is a path between a and b of length at most $t \times |ab|$, and therefore $SF_G(a,b) \leq t$. This proves the theorem. □

Theorem 4 *The graph G, generated by* GreedyPlaneSpanner*(S) is plane.*

Proof. According to the construction of G, any two edges (a,b) and (c,d) in G with $a,b,c,d \in \textbf{upper}$ or $a,b,c,d \in \textbf{lower}$, do not cross each other. Then, to prove the theorem, it is sufficient to prove that any two shortcut edges (a,b) and (c,d) in G do not cross each other. Suppose, for contradiction, that (a,b) and (c,d) cross each other. Suppose, without loss of generality, that the pair (a,b) is processed before the pair (c,d) by the algorithm ModifiedPathGreedy. Hence, $|ab| \leq |cd|$. Suppose, without loss of generality, that $a, c \in \textbf{upper}$ and $b, d \in \textbf{lower}$. Let u be the intersection of (a,b) and (c,d) (see Figure 1). Then, by the triangle inequality, we have $|ca| \leq |au| + |uc|$ and $|bd| \leq |bu| + |ud|$. Hence,

$$|ac| + |bd| \leq |au| + |uc| + |bu| + |ud| = |ab| + |cd| \leq 2|cd|. \quad (1)$$

Let P_{ca} be the convex path between c and a on $CH(S)$ using the points on **upper**, and let P_{bd} be the convex path between b and d on $CH(S)$ using the points **lower**. By Lemma 2, $|P_{ca}| \leq \frac{2\pi}{3}|ca|$ and $|P_{bd}| \leq \frac{2\pi}{3}|bd|$. Now, consider the path $Q := P_{ca} \cup (a,b) \cup P_{bd}$. Then,

$$\begin{aligned} |Q| &= |P_{ca}| + |ab| + |P_{bd}| \leq \frac{2\pi}{3}|ca| + |ab| + \frac{2\pi}{3}|bd| \\ &= \frac{2\pi}{3}(|ca| + |bd|) + |ab|. \end{aligned}$$

Since $|ab| \leq |cd|$ and by Equation (1), we have

$$\begin{aligned} |Q| &\leq \frac{2\pi}{3}(|ca| + |bd|) + |ab| \\ &\leq \frac{4\pi}{3}|cd| + |cd| = \frac{3+4\pi}{3}|cd| = t|cd|. \end{aligned}$$

Hence, the algorithm does not add the edge (c,d) to G, which is a contradiction. Then, (a,b) and (c,d) do not cross each other. Hence, G is plane. □

Now, we prove that the maximum degree of the graph G is at most 3.

Theorem 5 *The maximum degree of the graph G generated by* GreedyPlaneSpanner*(S) is at most 3.*

Proof. Let a be a point in S. We show that the degree of a in G is at most 3. Note that if $a = p$ or $a = q$, where (p,q) is the diametral pair of points which is selected by the algorithm, then since $CH(S) \subseteq G$ and $p, q \notin \textbf{upper} \cup \textbf{lower}$, clearly the degree of a is 2. Now, suppose that $a \neq p, q$. Suppose, without loss of generality, $a \in \textbf{upper}$. Since G contains $CH(S)$, then the degree of a is at least two. Suppose, for contradiction, that the degree of a is greater than 3. Hence, there exist two shortcut edges adjacent to point a. Now, suppose that (a,b) and (a,c) are two edges of G such that $b, c \in \textbf{lower}$ (see Figure 1). Suppose, without loss of generality, that the algorithm adds the edge (a,b) before the edge (a,c). Then, $|ab| \leq |ac|$. Now, let P_{bc} be the convex path between b and c on $CH(S)$ using the points of **lower**. By Lemma 2, we have $|P_{bc}| \leq \frac{2\pi}{3}|bc|$. Now, consider the path $Q := (a,b) \cup P_{bc}$. Now, we have $|Q| = |ab| + |P_{bc}| \leq |ac| + \frac{2\pi}{3}|bc|$. By the triangle inequality, we have $|bc| \leq |ab| + |ac| \leq 2|ac|$. Hence, by combining the two previous inequalities, we have $|Q| \leq \frac{3+4\pi}{3}|ac|$. Then, the algorithm does not add the edge (a,c) to G, which is a contradiction. Hence, the degree of a is at most 3. □

4 Time Complexity

We know that the running time of Dijkstra's single-source shortest paths algorithm for a weighted graph is $O(n \log n + m)$, where n is the number of vertices and m is the size of the graph. Moreover, sorting the list L in Algorithm 4 takes $O(n^2 \log n)$ time. Hence, a direct implementation of Algorithm 4 using Dijkstra's single-source shortest paths algorithm has the running

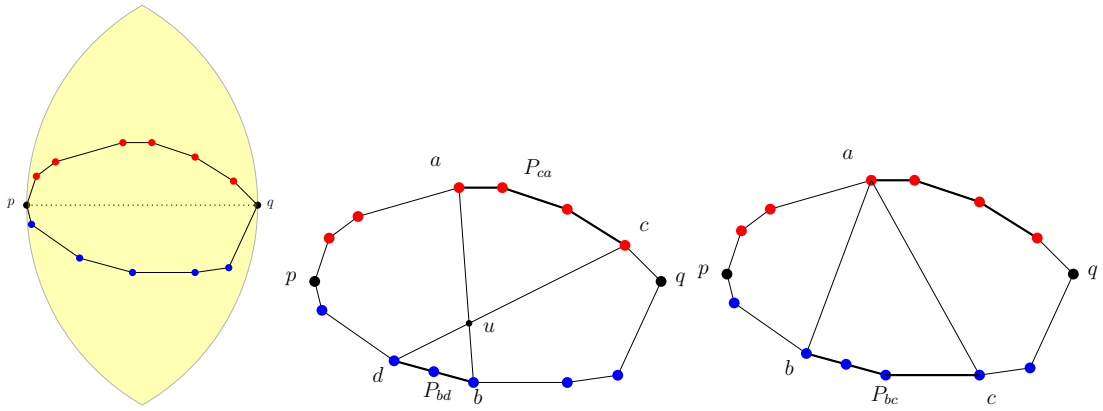


Figure 1: **(a)** The points with red color form U, the points with blue color form L and the yellow region is $L(p, q)$. **(b)**: Illustrating the proof of Theorem 4. **(c)**: Illustrating the proof of Theorem 5.

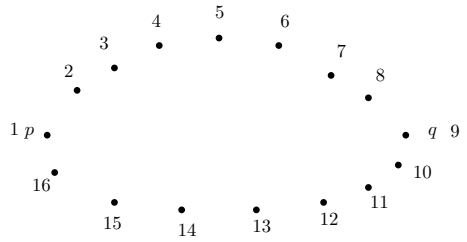


Figure 2: Numbering the point set S.

time $O(n^3 \log n)$. In this section, we show that the proposed plane spanner G computed by Algorithm 4 can be computed in $O(n^2 \log n)$ time.

The main idea to reduce the running time is that we not to use Dijkstra's single-source shortest paths algorithm, and instead by a quadratic-time preprocessing of S, we use an algorithm whose running time is $O(\log n)$ for each pair. Since there are $O(n^2)$ pairs, then the overall running time will be $O(n^2 \log n)$. Note that in [6] Bose et al. show how to compute the greedy spanner on a given point set in the plane in $O(n^2 \log n)$ time. We could not apply their algorithm here. We guess applying their algorithm would not give you the desired results. Now, we describe the algorithm in detail.

We number the points of S in the clockwise direction as depicted in Figure 2. Let x and y be the numbers assigned to p and q, respectively. Let T be a binary search tree (BST) that is initially empty. During the running of the algorithm, when we add an edge (i, j) to the graph, we add the numbers i and j to T. For two numbers $i, j \in$ **upper** (or $i, j \in$ **lower**) with $i < j$, let P_{ij} be the path from i to j on $CH(S)$ in the clockwise direction. Let A be an $n \times n$ array. For two numbers i and j with $i < j$ and $i, j \in$ **upper** (or $i, j \in$ **lower**), $A[i, j]$ is equal to the length of the path P_{ij} and for other values of i and j, $A[i, j]$ is equal to zero. Since the points of S are in convex position, it is not hard to see that the array A can be computed in $O(n^2)$ time. Now, we are ready to express the algorithm. The algorithm initially adds $CH(S)$ to the edge set E and computes the list L. Suppose that the algorithms want to process a pair (i, j) (i and j are numbers). Note that $i \in$ **upper** and $j \in$ **lower**. The algorithm searches in T to find the smallest number k, which is greater than i and the greatest number h, which is smaller than i. It is not hard to see that the numbers k and h can be computed in $O(\log n)$. Suppose that $k \neq \emptyset$ and $h \neq \emptyset$. Then, there exist two numbers $k', h' \in$ **lower** such that $(k, k') \in E$ and $(h, h') \in E$. Now, consider two paths $P := P_{ik} \cup (k, k') \cup P_{k'j}$ and $Q := P_{hi} \cup (h, h') \cup P_{h'j}$. Since the points are placed in convex position, one of two paths P and Q is the shortest path between i and j. Using the array A, we have $|P| = A[i, k] + |kk'| + A[k'j]$ and $|Q| = A[h, i] + |hh'| + A[h', j]$. Then, to determine whether the pair (i, j) should be added to the edge set E, it is sufficient to check if $|P| > t \times |ij|$ and $|Q| > t \times |ij|$. For the case $k = \emptyset$ and $h \neq \emptyset$, it is sufficient to consider $P := P_{iy} \cup P_{yj}$ and $Q := P_{hi} \cup (h, h') \cup P_{h'j}$. For the cases $k \neq \emptyset$ and $h = \emptyset$, and $k = \emptyset$ and $h = \emptyset$, the paths P and Q are defined similarly.

5 Weight of the Spanner

Now, we show that there is no upper bound on the total weight of the plane spanner G' proposed by Biniaz et al. [1], but the total weight of the plane spanner G proposed in the current paper is asymptotically equal to the $wt(MST(S))$.

Let S be a set of n points placed at vertices of a regular n-gon. Assume that n is sufficiently large. It is clear that the convex hull edges except one edge is a minimum spanning tree of S. Consider the plane span-

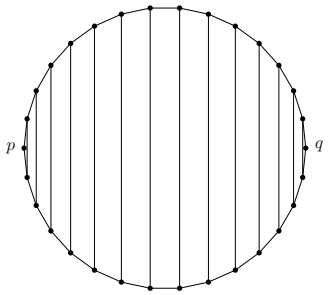


Figure 3: The plane spanner G' on a regular 30-gon.

ner G' on the point set S. According to Algorithm 3, the graph G' is similar to the one shown in Figure 3. Since G' contains many shortcut edges, it is clear that $\lim_{n\to\infty} wt(G') = \infty$. This shows that the weight of the proposed plane spanner by Biniaz et al. [1] is unbounded.

Now, we will analyze the total weight of the plane spanner G. First, the generalized leapfrog property is defined that we need later.

Definition 1 (Generalized Leapfrog Property [13]) *Let t_1 and t_2 be real numbers, such that $1 < t_1 < t_2$. A set E of undirected edges in $\mathbb{R}^d$ is said to satisfy the (t_1, t_2)-leapfrog property, if for every $\{p_1, q_1\}, \{p_2, q_2\}, \ldots, \{p_k, q_k\}$ of k pairwise distinct edges of E,*

$$t_1|p_1q_1| < \sum_{i=2}^{k} |p_iq_i| + t_2\left(|p_1p_2| + \sum_{i=2}^{k-1} |q_ip_{i+1}| + |q_kq_1|\right).$$

Now, we present the *Generalized Leapfrog Theorem*.

Theorem 6 (Generalized Leapfrog Theorem [13]) *There exists an absolute constant ϕ with $0 < \phi < 1$, such that the following holds. Let t_1 and t_2 be real numbers, such that $1 < 1 - \phi + \phi t_2 < t_1 < t_2$, let S be a set of points in $\mathbb{R}^d$, and let E be a set of edges, whose endpoints are from S, and that satisfies the (t_1, t_2)-leapfrog property. Then,*

$$wt(E) \leq c_{dt_1t_2} \cdot wt(MST(S)),$$

where $c_{dt_1t_2}$ is a real number that depends only on d, t_1, and t_2.

Given a graph; the length of the second shortest path between two vertices p and q in the graph is denoted by $\delta_2(p, q)$. If there is only one path between p and q in the graph, then we assume that $\delta_2(p, q) = \infty$. In the following, a sufficient condition for the leapfrog property is given.

Theorem 7 ([15]) *Let S be a set of n points in $\mathbb{R}^d$, let $t > 1$ be a real number, and let $G = (S, E)$ be an undirected t-spanner for S. Assume that $\delta_2(p, q) > t|pq|$, for every edge (p, q) in E. Then, the edge set E satisfies the (t, t^2)-leapfrog property.*

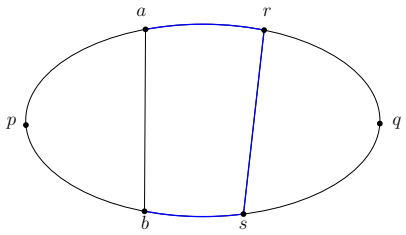


Figure 4: The path P with the gray color in the proof of Lemma 9.

Let $G = (S, E)$ be an undirected t-spanner for S and F be a subset of E. We easily verified the proof of Theorem 7 for the set F instead of E. Hence, we obtain the following result.

Theorem 8 *Let S be a set of n points in $\mathbb{R}^d$, let $t > 1$ be a real number, and let $G = (S, E)$ be an undirected t-spanner for S and F be a subset of E. Assume that $\delta_2(p, q) > t|pq|$, for every edge (p, q) in F. Then, the edge set F satisfies the (t, t^2)-leapfrog property.*

Let S be a set of points in the plane that is in convex position. Consider the plane spanner $G = (S, E)$ computed by the algorithm GreedyPlaneSpanner(S). Now, we prove the following result.

Lemma 9 *The set of all shortcut edges in G satisfies the $\left(\frac{3+4\pi}{3}, \left(\frac{3+4\pi}{3}\right)^2\right)$-leapfrog property.*

Proof. Let $t = \frac{3+4\pi}{3}$ and F is the set of all shortcut edges in G. By Theorem 8, to prove the lemma, it is sufficient to prove that for every edge $(a, b) \in F$, $\delta_2(a, b) > t|ab|$. Let P be a path between a and b in G having length $\delta_2(a, b)$.

Suppose that all edges of P are on $CH(S)$. Notice that, in this case, the path P passes through one of the diametral points. Since G initially has included the convex hull edges and (a, b) has been added to the edge set of G, according to the construction of G, $|P| > t|ab|$.

Now, suppose that one of the edges of P is not on $CH(S)$. It means that P contains a shortcut edge. Let C_1 and C_2 be two convex chains obtained by removing the points p and q from $CH(S)$. Since G is plane and the points of S are in convex position, the path P consists of some edges on C_1, a shortcut edge (r, s) and some edges on C_2 (see Figure 4). Suppose, without loss of generality, that $r \in C_1$ and $s \in C_2$. Note that since G is plane, it is not hard to see that in the quadrilateral $abrs$, there are no shortcut edges except two edges (a, b) and (r, s). Now, we show that $|P| > t|ab|$. If the algorithm adds the edge (r, s) before the edge (a, b), then according

to the construction of G, $|P| > t|ab|$. Now, suppose that (r, s) is added to G after the edge (a, b). Then, $|rs| \geq |ab|$. Let C_{ar} be the part of the path P that is on C_1 and C_{bs} be the part of P that is on C_2. Consider the path $Q := C_{ar} \cup (a, b) \cup C_{bs}$ between r and s. Since $|rs| \geq |ab|$, $|P| \geq |Q|$. Since (r, s) is an edge of G and Q is a path between r and s, which is created before processing the pair (r, s), $|Q| > t|rs|$. Hence,

$$|P| \geq |Q| > t|rs| \geq t|ab|.$$

This completes the proof. □

By Theorem 6 and Lemma 9, $wt(F) = O(1) \cdot wt(MST(\mathbf{lower} \cup \mathbf{upper}))$. Now, in the following we show that $wt(MST(\mathbf{lower} \cup \mathbf{upper})) \leq 2wt(MST(S))$.

Lemma 10 $wt(MST(\mathbf{lower} \cup \mathbf{upper})) \leq 2wt(MST(S))$.

Proof. Let $TSP(S)$ be the *traveling salesperson tour* on the point set S. It is not hard to see that $wt(TSP(\mathbf{lower} \cup \mathbf{upper})) \leq wt(TSP(S))$ (see Exercise 1.7 in [13]). Since $wt(MST(\mathbf{lower} \cup \mathbf{upper})) \leq wt(TSP(\mathbf{lower} \cup \mathbf{upper}))$, we have $wt(MST(\mathbf{lower} \cup \mathbf{upper})) \leq wt(TSP(S))$. On the other hand, it is well known that $wt(TSP(S)) \leq 2wt(MST(S))$ (see [13]). Hence, by combining two previous inequalities $wt(MST(\mathbf{lower} \cup \mathbf{upper})) \leq 2wt(MST(S))$. □

Now, we conclude this section with the following result.

Theorem 11 *For any set S of points in the plane that is in convex position, we have*

$$wt(G) = O(1) \cdot wt(MST(S)),$$

where $G = (S, E)$ is the plane spanner computed by GreedyPlaneSpanner*(S).*

Proof. Let F be the all shortcut edges in E. Let F' be the set of all edges on $CH(S)$. Clearly, $E = F \cup F'$. By Theorem 6, Lemma 9 and Lemma 10, we have $wt(F) = O(1) \cdot wt(MST(S))$. Now, since $TSP(S)$ only contains the convex hull edges (see [10]), we have $wt(F') = wt(TSP(S))$. Since $wt(TSP(S)) \leq 2wt(MST(S))$ (see [13]), $wt(F') \leq 2wt(MST(S))$. Now, we conclude that $wt(E) = wt(F) + wt(F') = O(1) \cdot wt(MST(S))$. □

6 Conclusion

In this paper, we presented an algorithm that constructs a plane $\frac{3+4\pi}{3}$-spanner of degree at most three in $O(n^2 \log n)$ time for any set of n points in the plane that is in convex position. We showed that the total weight of the proposed plane spanner is asymptotically equal to the total weight of the minimum spanning tree of the points.

We conclude the paper with the following open problems.

Figure 5: The point set P.

1. *Is the size (number of the edges) of the proposed plane spanner G less than or equal to the size of the plane spanner G' proposed by Biniaz et al. [1]?*

   To tackle this problem, one direction is to show that the graph G is a subgraph of G'. We found the following counterexample for this. Assume the points of P placed on the sides of the rectangle $pdqe$ such that $|qd| = |cd| = |pe| = 1$, $|dc| = |qb| = \frac{2\pi}{3}$ and $|ab| = 0.4$ (see Figure 5). It is not hard to see that the graph G contains the edge (a, c), but G' does not contain (a, c) and contains the edge (b, c) instead.

2. *Does the algorithm* PathGreedy*$(S, \frac{3+4\pi}{3})$ output a plane spanner of degree at most three for any set S of points in the plane that is in convex position?*

   We guess this problem has a yes-answer. We tried to show this, but we did not succeed.

Acknowledgment

Some parts of this work have been done at the *Open Problem Session*, held at Yazd University, 13-15 July 2019. The first author would like to thank Mohammad Farshi for his warm hospitality at Yazd.

References

[1] A. Biniaz, P. Bose, J.-L. De Carufel, C. Gavoille, A. Maheshwari, and M. Smid. Towards plane spanners of degree 3. *Journal of Computational Geometry*, 8(1):11–31, 2017.

[2] A. Biniaz and M. Smid. *Personal Communication*, 2019.

[3] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perković. Plane spanners of maximum degree six. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *Automata, Languages and Programming*, pages 19–30, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[4] N. Bonichon, I. Kanj, L. Perković, and G. Xia. There are plane spanners of degree 4 and moderate stretch factor. *Discrete & Computational Geometry*, 53(3):514–546, Apr 2015.

[5] P. Bose, P. Carmi, and L. Chaitman-Yerushalmi. On bounded degree plane strong geometric spanners. *Journal of Discrete Algorithms*, 15:16 – 31, 2012.

[6] P. Bose, P. Carmi, M. Farshi, A. Maheshwari, and M. Smid. Computing the greedy spanner in near-quadratic time. *Algorithmica*, 58(3):711–729, Nov 2010.

[7] P. Bose, J. Gudmundsson, and M. Smid. Constructing plane spanners of bounded degree and lowweight. *Algorithmica*, 42(3):249–264, Jul 2005.

[8] P. Bose, M. Smid, and D. Xu. Delaunay and diamond triangulations contain spanners of bounded degree. *International Journal of Computational Geometry & Applications*, 19(02):119–140, 2009.

[9] G. Das and P. J. Heffernan. Constructing degree-3 spanners with other sparseness properties. *International Journal of Foundations of Computer Science*, 07(02):121–135, 1996.

[10] V. G. Deĭneko, M. Hoffmann, Y. Okamoto, and G. J. Woeginger. The traveling salesman problem with few inner points. *Operations Research Letters*, 34(1):106 – 110, 2006.

[11] I. Kanj, L. Perković, and D. Turkoglu. Degree four plane spanners: Simpler and better. *Journal of Computational Geometry*, 8(2):3–31, 2017.

[12] X.-Y. Li and Y. Wang. Efficient construction of low weighted bounded degree planar spanner. *International Journal of Computational Geometry & Applications*, 14(01n02):69–84, 2004.

[13] G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[14] L. Perković and I. A. Kanj. On geometric spanners of euclidean and unit disk graphs. In *25th International Symposium on Theoretical Aspects of Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[15] M. Smid. Errata for the book geometric spanner networks. 2013.

Appendix

Algorithm 1: PathGreedy(S, t)

input: a set S of n points in $\mathbb{R}^d$ and a real number $t > 1$.
output: t-spanner $G(S, E)$.

Sort $\binom{n}{2}$ pairs of points in non-decreasing order of their distances (ties are broken arbitrarily), and store them in list L;
$E := \emptyset$;
$G := (S, E)$;
foreach pair $(u, v) \in L$ (in sorted order) **do**
  if ShortestPath$(G, u, v) > t \cdot |uv|$ **then**
    $E := E \cup \{(u, v)\}$;
  end
end
return $G(S, E)$;

Algorithm 2: ModifiedPathGreedy(S, E, L, t)

$G := (S, E)$;
foreach pair $(u, v) \in L$ (in sorted order) **do**
  if ShortestPath$(G, u, v) > t \cdot |uv|$ **then**
    $E := E \cup \{(u, v)\}$;
  end
end
return $G(S, E)$;

Non-Crossing Matching of Online Points

Prosenjit Bose* Paz Carmi† Stephane Durocher‡ Shahin Kamali‡ Arezoo Sajadpour‡

Abstract

We consider the non-crossing matching problem in the online setting. In the monochromatic setting, a sequence of points in general position in the plane is revealed in an online manner, and the goal is to create a maximum matching of these points such that the line segments connecting pairs of matched points do not cross. The problem is online in the sense that the decisions to match each arriving point are irrevocable and should be taken without prior knowledge about forthcoming points. The bichromatic setting is defined similarly, except that half of the points are red and the rest are blue, and each matched pair consists of one red point and one blue point. Inspired by the online bipartite matching problem [15], where vertices on one side of a bipartite graph appear in an online manner, we assume red points are given a priory and blue points arrive in an online manner.

In the offline setting, both the monochromatic and bichromatic problems can be solved optimally with all pairs matched [11]. In the online setting of the monochromatic version, we show that a greedy family of algorithms matches $2\lceil (n-1)/3 \rceil$ points, where n is the number of input points. Meanwhile, we prove that no deterministic online algorithm can match more than $2\lceil (n-1)/3 \rceil$ points, i.e., the greedy strategy is optimal. In the bichromatic setting, we introduce an algorithm that matches $\log n - o(\log n)$ points for instances consisting of n red and n blue points, and show that no deterministic algorithm can do better. We also consider the problem under the advice setting, where an online algorithm receives some bits of advice about the input sequence, and provide lower and upper bounds for the amount of advice that is required and sufficient to match all points.

1 Introduction

Matching is an important topic in combinatorics, particularly in graph theory (see, e.g., the book by Lovász and Plummer [17]). When it comes to computational geometry, matching of points in the plane has applications that range from circuit design [12] to colour-based image retrieval [1]. In a monochromatic setting, a collection of points in general position is given and the goal is to match a maximum number of points, provided they adhere to some constraints. In the bichromatic variant, each point is either blue or red, and must be matched to a point of the opposite color. Variants of the problems have been considered. For example, one might be interested in minimizing the total length or the maximum length of segments in the matching, also known as bottleneck matching (e.g., [19, 8]). Kaneko and Kano survey some of the results related to this setting [14].

In the *non-crossing matching problem*, the goal is to find a maximum matching so that the segments between pairs of matched points do not intersect. In the offline setting, where all points are given as input in advance, the problem can be easily solved in both the monochromatic and the bichromatic settings. In the case of monochromatic points, one can sort them by their x-coordinate and match consecutive pairs in the sorted sequence. This will match all points except potentially the last one (if the number of points is odd). For the bichromatic variant [2], also known as the "Ghosts and Ghostbusters" problem [7], one can find the ham-sandwich line that bisects the blue and red points in $O(n)$ time [16], and apply a divide-and-conquer approach. Both algorithms run in $O(n \log n)$ time for an input of size n, which is best possible [9]. Assuming the number of red and blue points are equal, all points are matched. A slightly different approach, with the same running time, is presented in [11]. We also note that a minimum-length matching is noncrossing. In summary, we can match all points optimally in $O(n \log n)$ time in the offline setting. Other variants of non-crossing matching have been studied (see [17]). For example, Aloupis et al. [1] considered the computational complexity of finding non-crossing matching of a set of points with a set of geometric objects, where an object can be a convex polygon, a line, or a line segment.

In this article, we are interested in the online variant of non-crossing matching problems.

Definition 1 *The input to the* ***monochromatic online non-crossing matching problem*** *is a set of points in general position in the plane that appear in an online, sequential manner. When a point arrives,*

*Carleton University, Ottawa, Canada. jit@scs.carleton.ca

†Ben-Gurion University of the Negev, Beer-Sheva, Israel. carmip@cs.bgu.ac.il

‡University of Manitoba, Winnipeg, Canada. {durocher,shahin.kamali}@cs.umanitoba.ca, sajadpoa@myumanitoba.ca

an online algorithm can match it with an existing unmatched point, provided that the line segment between them does not cross previous line segments added to the matching. Alternatively, the algorithm can leave the point unmatched to be matched later. In taking these decisions, the algorithm has no information about the forthcoming points or the length of the input. The algorithm's decisions are irrevocable in the sense that once a pair of points is matched, that pair cannot subsequently be removed from the matching. The objective is to find a maximum matching. In the ***bichromatic variant*** *of the problem, half of the points are red and half are blue. The red points are given in advance, while the blue points appear in an online manner. Upon arrival of a blue point, an online algorithm either matches it with a red point or leaves it unmatched. The goal is to find a maximum matching in which the line segments between matched pairs do not cross.*

In the online setting, it is not always possible to match all points to achieve an optimal solution. As an example, consider two points with the same x-coordinate appear at the beginning. If the online algorithm does not pair them, its solution is sub-optimal for an input formed only by these two points. If the algorithms does pair the first two points, the sequence might be followed by one point on the left and one on the right of the line segment between the matched pair. The new points cannot be matched and hence the solution is sub-optimal for an input formed by the four points.

We study the online matching problem in the worst-case scenario, where the input is generated by an adversary. This is consistent with the standard framework of competitive analysis [18]. The *competitive ratio* of an online algorithm is the maximum ratio between the number of pairs in an optimal offline solution and that of the online algorithm for sufficiently long sequences. Since an offline algorithm always matches all points (except potentially one), we prefer to express our results in terms of the number of matched/unmatched points. Throughout the paper, we assume the length n of the online sequence is sufficiently large.

1.1 Contribution

For the monochromatic variant of the problem, we consider greedy algorithms with the following *greedy property*: the algorithm never leaves an incoming point unmatched if it can be matched with some existing point. We prove that a greedy algorithm can match at least $\lceil 2(n-1)/3 \rceil$ points for any input of n points. Moreover, we prove optimality since no deterministic algorithm can match more than $\lceil 2(n-1)/3 \rceil$ points in the worst case.

For the bichromatic variant, we introduce an algorithm that matches at least $\log n - o(\log n)$ points for any input formed by n red and n blue points. Further, we prove optimality since no deterministic algorithm can match more points in the worst case. Our results indicate that the bichromatic variant is more difficult than the monochromatic variant in the online setting.

In addition to the purely online setting, we study the problem in a relaxed setting where the online algorithm is provided with some bits of *advice* about the input. The advice is generated by an offline oracle, and is available to the algorithm before the sequence is revealed (see [3, 5, 6] for a precise definition of advice). For the monochromatic variant, we show that advice of size $2n$ is sufficient to match all n points, and advice of size $\lfloor \log((n-2)/3) \rfloor$ is necessary. For the bichromatic variant, we show advice of size $\Theta(n \log n)$ is both sufficient and necessary to match all points; precisely $n\lceil \log n \rceil$ bits are sufficient and $\lceil \log n! \rceil$ bits are necessary.

2 Monochromatic Non-crossing Matching

In this section, we provide tight upper and lower bounds for the number of points that can be matched in the monochromatic non-crossing matching problem.

An online algorithm is said to have the *greedy property* if it never leaves a point unmatched when it has the option to match it.

Theorem 1 *Any online algorithm with the greedy property matches at least $2\lceil (n-1)/3 \rceil$ points in any instance of the online monochromatic non-crossing matching problem on n points.*

Proof. Let Gr be a greedy algorithm. The proof works by partitioning the plane into a set of convex regions such that each region, except one, is mapped to a pair of matched vertices. For that, we process the line segments between matched pairs of Gr in an arbitrary order. Initially, there is only one part, formed by a bounding box of the entire point set; this part has no pair associated with it. Extend each line segment until it intersects an existing line in the current partition. Note that the extended segment divides one convex region into two smaller convex regions, out of which we associate one with the pair that has been processed, and the other to the pair that was previously associated with the partitioned convex region. Repeating this process for all line segments results in $k+1$ convex regions in the final partition, where k is the number of matched pairs (see Figure 1). For detailed geometric properties of this convex subdivision, see, e.g., [4, 13]. Since Gr has the greedy property, there is at most one unmatched point inside each convex region.

To summarize, the number of unmatched points u is no more than the number of convex regions, which is one more than the number of matched pairs m. So, we

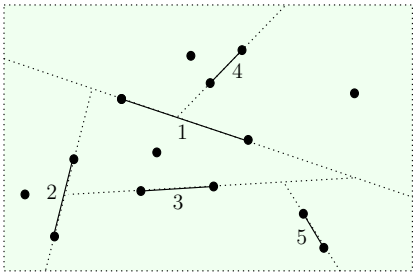


Figure 1: A partition of the plane into convex regions in the analysis of a greedy algorithm. The numbers on the line segments indicate the order they are processed in the analysis.

have $u \leq m + 1$. The statement follows from the fact that $u + 2m = n$. □

Next, we show that no online algorithm matches more points than does a greedy algorithm in the worst case.

Theorem 2 *Let* Alg *be any deterministic online algorithm for the monochromatic non-crossing matching problem. There are sequences of n points for which* Alg *matches at most $2\lceil (n-1)/3 \rceil$ points.*

Proof. We form an input that is generated in an adversarial manner based on the actions of Alg. The adversary maintains a *critical region*, which is initially the entire plane and *shrinks* as the algorithm proceeds. The adversary keeps adding points to arbitrary positions in the critical region. As soon as the algorithm matches two points a and b, the critical region is updated as follows. Consider the two sides of the line passing through a and b. If there is a non-empty set S of unmatched points on any side of the line in the critical region, then the critical region is updated to be its sub-region that is not *visible* to any point in $x \in S$ assuming the line segment between a and b acts as an obstacle. This can be done by extending the line segments between x and a and b (see Figure 2). Since the points are in general position, the updated critical region is non-empty. Note that if both sides of the line passing through a and b include unmatched points, the adversary selects one side arbitrarily. In case no unmatched point exists in the critical region, the adversary first generates a point x in an arbitrary position in the critical region and updates the critical region as a sub-region not visible by x. This process continues by sending the subsequent points in the updated (smaller) critical region.

The main observation is that, after a critical region is updated, at least one point x remains unmatched since the line segment between x and any future point crosses the segment between a and b. In particular, we can assign at least one unmatched point x to a matched pair. After updating the critical region, the very first point generated in the updated region also remains unmatched. Let u and m denote the number of unmatched points and matched pairs, respectively. By the above observations, we have $u = m + 1$. The statement of the theorem follows from $u + 2m = n$. □

3 Bichromatic Non-crossing Matching

In this section, we study online algorithms for the bichromatic non-crossing matching problem and provide tight upper and lower bounds for the number of points that can be matched. Recall that the input is formed by n red points that are known to the algorithm from the beginning and n blue points that appear in an online manner and need to be matched with the red points.

We introduce an online algorithm, named the *Greedy Median* (Gm) algorithm, that works as follows. Upon arrival of a blue point a, Gm forms a set S of eligible red points that can be matched with a without crossing previous line segments. If S is non-empty, Gm matches a with the median of the points in S when arranged in angular order around a. The selection of angular ordering is arbitrary, and it can be replaced by any ordering as long as the line through a and the median of the points in S bisects S.

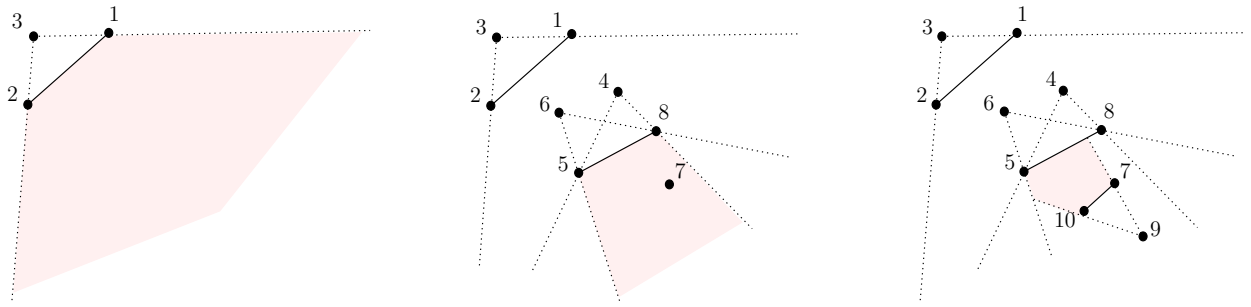
Theorem 3 *The Greedy Median (Gm) algorithm matches at least $\log(n) - o(\log n)$ pairs of points in any instance of the bichromatic non-crossing matching problem formed by a set of n red points and a sequence of n blue points.*

Proof. Let $M(n)$ denote the number of matched pairs by Gm in the worst case in an instance formed by n blue and n red points (we have $M(1) = 1$). The algorithm matches the first blue point with the median of the red points. Consider the two sides of the line that passes through the matched pair. One of the two sides contains at least half of the future blue points, i.e., at least $\lfloor (n-1)/2 \rfloor$ blue points. There are also $\lfloor (n-1)/2 \rfloor$ red points on the same side (since the line bisects the red points). So, we have $M(n) \geq 1 + M(\lfloor \frac{n-1}{2} \rfloor)$ for $n > 1$, which solves to $M(n) \geq \log(n) - o(\log n)$. □

Although it is not difficult to match $\log n - o(\log n)$ points, as we now show, no online algorithm can guarantee to match more than $\log n - o(\log n)$ points.

Theorem 4 *Let* Alg *be any deterministic online algorithm for the bichromatic non-crossing matching problem. There are inputs formed by a fixed set of n red points and a sequence of n blue points for which* Alg *matches at most $\log n - o(\log n)$ points.*

Proof. We create an adversarial input in which n red points are placed in arbitrary positions on an arc of a



(a) (b) (c)

Figure 2: An illustration of updating the critical region (pink region) by the adversary in the proof of Theorem 2. The numbers on the points indicate their index in the input sequence. (a) Once points 1 and 2 are matched, there is no unmatched point in the critical region; the adversary generates point 3 and updates the critical region to its subregion that is not visible to 3. (b) Assume the algorithm does not match the next points 4, 5, 6, and 7. When it matches points 5 and 8, points in $S = \{4, 6\}$ are unmatched on one side of the line passing through 5 and 8. The adversary updates the critical region to be its subregion not visible by any member of S. (c) Assume the algorithm does not match the next point 9. When it matches points 10 and 7, the set $S = \{9\}$ is unmatched on one side of the line. The critical region is updated to be its subregion not visible to 9.

large circle so that they seem collinear except that the corresponding arc slightly curves outwards. The blue points appear in an online manner below the red point on a similar arc that slightly curves inwards; this arc is referred to as a *critical region* at the beginning, and is updated as the algorithm matches points. Assume at some point ALG matches an incoming blue point with a red point, and let L be the line that passes through the matched pair. The number of red points on one side of L is at most $\lfloor (n-1)/2 \rfloor$. The adversary updates the critical region to only include this side of L. This ensures that at least $\lceil (n-1)/2 \rceil$ red points on the other side of L remain unmatched; this is because the line segments between these points and all future blue points (generated in the updated critical region) crosses L (see Figure 3). So, each time ALG matches two points, the number of red points that can still be matched decreases by a factor of at least 2. Consequently, the number of matched pairs is at most $\log n - o(\log n)$. □

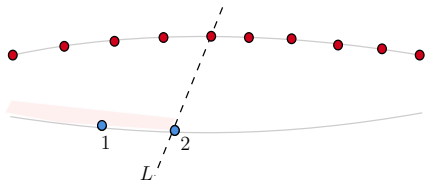


Figure 3: Updating the critical region by the adversary in the proof of Theorem 4. In the beginning, the critical region is the entire lower arc. Assume ALG does not match the first blue point but the second one is matched. The majority of red points appear on the right of the line L passing through the matched pair. As such, the adversary updates the critical region to be the left of L.

4 Non-Crossing Matching with Advice

In this section, we study the non-crossing matching problem under the advice model. We refer the reader to [5] for a survey on online algorithms with advice. Under the advice model, an online algorithm is provided with some bits of advice about the input sequence. The advice can encode any information about the input sequence, and is generated by a benevolent offline oracle that knows the entire input. A central question under the advice model asks for the number of advice bits necessary/sufficient to achieve an optimal solution. In the context of the non-crossing matching problem, this question translates to the number of advice bits needed to match all points.

4.1 Monochromatic setting

In this section, we study the monochromatic non-crossing matching problem under the advice setting. First, we show that $O(n)$ bits of advice is sufficient to match all the points.

Theorem 5 *There is an online algorithm that receives* $(\log_2 3)\ n + o(n) \leq 1.59n$ *bits of advice and matches all points (except one if* n *is odd) in any instance of the online monochromatic non-crossing matching problem on* n *points.*

Proof. Consider an offline matching that sorts the points by their x-coordinate and matches consecutive pairs of points. Call these pairs of matched points "partners". Note that all points are matched by this offline algorithm (except one if n is odd). Now, for each point p, we generate an advice $f(p) \in \{0, 1, 2\}$, based on this offline matching, as follows:

- when the partner of p appears after p in the online sequence, we define $f(p) = 0$.
- when the partner of p appears before p and is located to the left of p, we define $f(p) = 1$.
- when the partner of p appears before p and is located to the right of p, we define $f(p) = 2$.

So, the advice forms a string of length n over an alphabet of size 3. This can be encoded in $(\log_2 3)\ n + o(n) < 1.59n$ bits using, e.g., a wavelet tree structure [10].

It remains to show how to match points using the advice. Assume a point p arrives. If the advice encoded for p is 0, the algorithm keeps it unmatched as its partner has not arrived yet. If the advice is 1 or 2, then, p should be matched with the point with closest x-coordinate on its left or right, respectively. Using this scheme, we obtain a matching that is the same as the optimal offline solution. □

Lower Bound. In what follows, we show that advice of size $\Omega(\log n)$ bits is required in order to match all points in a given sequence of n points (assume n is even). Our lower bound argument generates sequences in which all points are on the circumference of a circle. In the offline setting, we can index the points, in clockwise order, starting from an arbitrary position. Any matching of a point with an even index to a point with an odd index divides the problem into two even-sized sub-problems, which can be solved recursively. Any such matching is equivalent to a balanced parenthesis sequence (see Figure 4). Consequently, in the offline setting, there are $C_{n/2}$ different ways to match all points, where $C_{n/2}$ is the $(n/2)$th Catalan number.

In order to provide a lower bound for the size of advice bits required to match all points, we create a family of $n-2$ input sequences of length n, denoted by $\sigma_1, \sigma_2, \ldots, \sigma_{n-2}$. All these sequences start with a common prefix $p_1, p_2, \ldots, p_{n-2}$, where the p_i's appear in clockwise order on the circumference of a circle. The last two points of any sequence σ_i are x_i and y_i, where x_i is a point located between p_{i-1} and p_i, and y_i is a point located between p_i and p_{i+1}.

Assume an online algorithm Alg (with advice) is applied on a sequence σ_i. Define a *partial matching* as the (incomplete) solution of Alg for the common prefix of the sequences in the family (the first $n-2$ points). In the partial solution, some points are matched, call them *partners*, and some are unmatched. A partial matching is said to be *valid* for σ_i, iff it can be completed such that all points in σ_i are matched at the end.

Lemma 6 *Any partial matching is valid for at most two sequences from the family.*

Proof. A valid partial matching for any sequence in the family should have exactly two unmatched points.

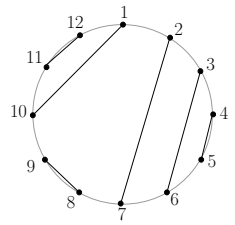


Figure 4: When points are located on the circumference of a circle, an offline algorithm can match all points by matching even-indexed points with odd-indexed points. The parentheses sequence associated with this matching is $(_1\ (_2\ (_3\ (_4\ )_5\ )_6\ )_7\ (_8\ )_9\ )_{10}\ (_{11}\ )_{12}$.

If more than two points are unmatched, some will stay unmatched at the end since only two more points from each sequence is left. If all points are matched, the last two points x_i and y_i in σ_i remain unmatched since the line segment between them crosses the line segment between p_i and its partner. So, we can consider a partial matching $S_{i,j}$ where two points p_i and p_j are unmatched. There are two cases to consider:

Case I: assume the line segment between p_i and p_j does not cross any line segment between matched pairs in $S_{i,j}$. We claim $S_{i,j}$ cannot be valid for any σ_k, where $k \notin \{i, j\}$. Consider a line L passing through p_k and its partner $p_{k'}$ in $S_{i,j}$. Both p_i and p_j appear on the same side of L. Among x_k and y_k, one appears on the same side of L while the other appears on the other side. In short, three unmatched points appear on one side of L and one on the other side (see Figure 5a). We cannot match all points without crossing L.

Case II: assume the line segment between p_i and p_j crosses a line segment L between p_k and its partner $p_{k'}$. So, p_i and p_j appear on different sides of L, which implies the remaining two points should be also on different sides of L to be matched with p_i and p_j. This is only possible for σ_k and $\sigma_{k'}$ (see Figure 5b). Note that if the line segment between p_i and p_j crosses more than one line segment in $S_{i,j}$, the same argument implies that the remaining points should be on the two sides of two existing line segments in $S_{i,j}$ at the same time, which is not possible (see Figure 5c). □

Using Lemma 6, we can prove the following lower bound on the size of advice required to match all points.

Theorem 7 *A deterministic algorithm requires advice of size at least $\lfloor \log((n-2)/3) \rfloor$ in order to guarantee matching all points in any instance of the online monochromatic non-crossing matching problem on n points.*

Proof. Assume, for the sake of a contradiction, that there is an algorithm Alg that matches all points

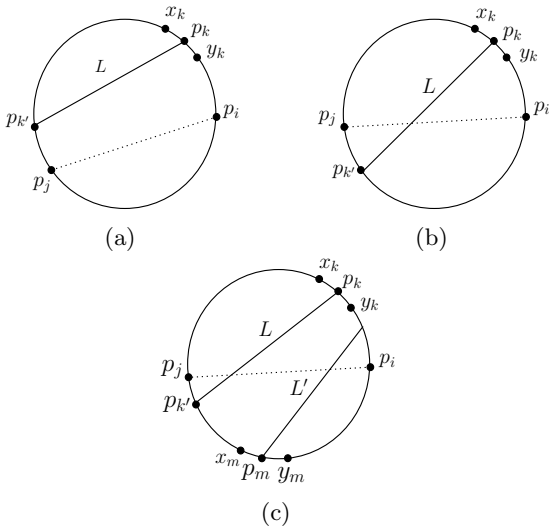


Figure 5: An illustration of Lemma 6. We have a partial matching $S_{i,j}$ where all points except p_i and p_j are matched. (a) if the line segment between p_i and p_j does not cross existing segments in $S_{i,j}$, it is not possible to match all points of any sequence σ_k for $k \notin \{i, j\}$. (b) if the line passing through p_i and p_j crosses a line segment between two matched points p_k and $p_{k'}$, then it might be possible to match the remaining points of σ_k and $\sigma_{k'}$. (c) if the line segment passing through p_i and p_j crosses two line segments L and L' between matched points, we cannot match the remaining two points.

in any instance of length n with less than $\alpha(n) = \lfloor \log((n-2)/3) \rfloor$ bits of advice. In particular, ALG should match all points for any sequence in the family $\{\sigma_1, \ldots, \sigma_{n-2}\}$ as we described above. We partition this set into $2^{\alpha(n)} \leq (n-2)/3$ *sub-families*, each formed by sequences that receive the same advice bits. Since there are $n-2$ sequences and at most $(n-2)/3$ sub-families, there is a sub-family with at least 3 sequences, that is, there are three sequences σ_a, σ_b, and σ_c that receive the same advice. Since these three sequences have the same common prefix and receive the same advice, ALG treats them similarly for the first $n-2$ points. That is, the partial matching of ALG is the same for all σ_a, σ_b, and σ_c. By Lemma 6, however, this partial matching is not valid for at least one of these sequences. We conclude that ALG cannot match all points for at least one sequence, a contradiction. □

4.2 Bichromatic setting

We show that advice of size $\Theta(n \log n)$ is both sufficient and necessary to match all points in the bichromatic setting. The more complicated nature of the bichromatic setting implies that advice of size of $\Theta(n)$ is insufficient (unlike the monochromatics setting) and, at the same time, simplifies our lower and upper bound arguments.

Theorem 8 *Consider any instance of the online bichromatic non-crossing matching problem with a sequence of n blue and a fixed set of n red points. There is a deterministic algorithm that receives $n\lceil \log n \rceil$ bits of advice and matches all points. Meanwhile, any deterministic algorithm requires advice of size at least $\lceil \log n! \rceil$ bits in order to match all points.*

Proof.
Upper bound: The offline oracle creates an ordering of the red points (say ordered by x-coordinate and ties broken by y-coordinate) and computes an optimal biochromatic matching on these. Now, for each blue point x, it encodes an advice of size $\lceil \log n \rceil$ that indicates the label of the red point to which x is matched. The online algorithm can mimic the offline matching by forming the same ordering of red points and matching each blue point to the red point indicated in the advice.
Lower bound: Consider instances of the problem in which the n red points $r_1, r_2, \ldots, r_n$ are placed, from left to right, on an arc of a large circle so that they seem collinear. The blue points $b_1, b_2, \ldots, b_n$ appear below the red points on an arc that slightly curves inwards (similar to Figure 3). In order to match all points, the left-most red point (r_1) should be matched with the left-most blue point (b_1). Using an inductive argument, we can show there is a unique matching of all points, where r_i is matched with b_i. Consider a family of $n!$ sequences, each associated with a permutation of the blue points $b_1, \ldots, b_n$ that indicates the order at which they appear in the online sequence. Let ALG be a deterministic online algorithm with less than $\lceil \log n! \rceil$ bits of advice. This implies that two sequences σ and σ' in the family receive the same advice. Assume the permutations associated with σ and σ' differ for the first time at index i, and let x be the i'th point in the input sequence. In σ, the point x is b_k and in σ' it is $b_{k'}$ for some $k \neq k'$. Since ALG is deterministic and receives the same advice for σ and σ', it matches x with the same red point in both cases. Such a matching, however, is not consistent with the unique optimal matching for at least one of the two sequences. As such some points remain unmatched in either σ or σ', and hence ALG fails to match all points. □

5 Concluding Remarks

Theorems 5 and 7 indicate that advice of size $O(n)$ and $\Omega(\log n)$ are respectively sufficient and necessary to match all points in the monochromatic setting. Closing the gap between these bounds does not seem to be easy and requires alternative techniques.

All algorithms studied in this paper are deterministic. We expect that randomization can improve the expected number of matched points which we propose as a direction for future research.

Acknowledgement

We thank the anonymous reviewers for their useful suggestions. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). NSERC funds were used for the research visit that resulted in publication of this paper.

References

[1] G. Aloupis, J. Cardinal, S. Collette, E. D. Demaine, M. L. Demaine, M. Dulieu, R. F. Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara, and P. Taslakian. Non-crossing matchings of points with geometric objects. *Comput. Geom.*, 46(1):78–92, 2013.

[2] M. J. Atallah. A matching problem in the plane. *J. Comput. Syst. Sci.*, 31(1):63–70, 1985.

[3] H. Böckenhauer, D. Komm, R. Královic, and R. Královic. On the advice complexity of the k-server problem. *J. Comput. Syst. Sci.*, 86:159–170, 2017.

[4] P. Bose, M. E. Houle, and G. T. Toussaint. Every set of disjoint line segments admits a binary tree. *Discret. Comput. Geom.*, 26(3):387–410, 2001.

[5] J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.

[6] J. Boyar, S. Kamali, K. S. Larsen, and A. López-Ortiz. Online bin packing with advice. *Algorithmica*, 74(1):507–527, 2016.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[8] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[9] J. Erickson. https://mathoverflow.net/questions/86906. https://stackexchange.com/. Accessed: 2020-07-21.

[10] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th Symp. on Discrete Algorithms (SODA)*, pages 841–850, 2003.

[11] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Comput. Sci. Sect.*, 32(2):249–267, 1992.

[12] J. Hershberger and S. Suri. Efficient breakout routing in printed circuit boards. In J. Boissonnat, editor, *Proc. 13th Annual Symposium on Computational Geometry (SOCG)*, pages 460–462. ACM, 1997.

[13] M. Hoffmann, B. Speckmann, and C. D. Tóth. Pointed binary encompassing trees: Simple and optimal. *Comput. Geom.*, 43(1):35–41, 2010.

[14] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane—a survey. *Discrete & Computational Geometry*, 25:551–570, 2003.

[15] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In H. Ortiz, editor, *STOC90*, pages 352–358. ACM, 1990.

[16] C. Lo, J. Matousek, and W. L. Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11:433–452, 1994.

[17] L. Lovász and M. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. North-Holland, 2009.

[18] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.

[19] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.

Restricted-Weight Minimum-Dilation Spanners on Three Points

Kevin Buchin* Herman Haverkort† Hidde Koerts

Abstract

Given a planar point set P and a parameter $L > 0$, we are interested in finding a Euclidean graph G, possibly using Steiner vertices, that has total weight at most L and minimizes the maximum dilation between any pair of points $p, q \in P$. The dilation between two points is the ratio between their graph distance and their Euclidean distance.

While this problem can be approached using convex optimization, the geometry of solutions is not yet well understood. We investigate this problem for the case $P = \{A, B, C\}$ of three points. In this case the solution consists of a triangle $\Delta A'B'C'$ and edges AA', BB' and CC'. We show that if A, B and C are the vertices of an equilateral triangle ΔABC, then $\Delta A'B'C'$ is equilateral and centered in ΔABC with its vertices on the bisectors of $\angle ABC, \angle ACB, \angle BAC$. We further analyze the solution for the case that ΔABC is isosceles and L is small.

1 Introduction

A Euclidean graph is a graph in which each vertex is a point in $\mathbb{R}^n$, and each edge (p, q) has weight equal to the Euclidean distance $|pq|$ between p and q. Let $G(V, E)$ be a Euclidean graph and let $d_G(p, q)$ denote the shortest distance between $p, q \in V$ in graph G, that is, the smallest total weight of any path from p to q. Let furthermore $D_G(p, q) = \frac{d_G(p,q)}{|pq|}$ be the dilation between $p, q \in V$ over graph G. A *spanner* of a set of points $P \in \mathbb{R}^n$ is then defined as a connected graph $G = (V, E)$ such that $P \subset V$. Depending on the context, one may require $V = P$, or one may allow V to contain *Steiner points*, that is, points that are not in P. The dilation D_G of the spanner is its maximum dilation over all pairs of points from P, that is, $D_G = \max_{p,q \in P} D_G(p, q)$. A *t-spanner* is a spanner G with dilation $D_G \leq t$. Spanners and related algorithms have extensively been described by Narasimhan and Smid [15].

Spanners form an important concept in computational geometry. Geometric spanners have various applications in for example the searching of metric spaces [16], the distribution of messages in networks [10], and the creation of approximate distance oracles [11]. Generally, geometric spanners can be utilized to approximate more complex networks, allowing for more time-efficient approximation algorithms. In the theoretical analysis of such algorithms, dilation is often used to prove bounds on the accuracy of said algorithms.

Spanners are frequently studied under further constraints regarding a variety of measures. Commonly considered measures include the number of edges in the spanner, diameter of the spanner and the total weight of the spanner. The basic spanner problem concerns minimizing the number of edges for a given desired dilation [15]. The greedy spanner, as introduced by Althöfer et al. [2], has been proven asymptotically optimal for the basic spanner problem [14]. Further approximation algorithms aim for asymptotic complexities of $O(n)$ and $O(|MST|)$ for the number of edges and the weight respectively [1, 3, 4, 5, 15]. An Integer Linear Program formulation for the basic spanner problem, was given by Sigurd and Zachariasen [17]. These results are for the setting without Steiner vertices.

The knowledge on other settings is less extensive. In this report, we investigate spanners with Steiner vertices and constrained in weight and with minimal dilation. The related Minimum-Weight Spanner Problem (without Steiner points), considering the minimal weight for a given dilation, has been proven to be NP-hard [7]. Generally, little is known about the exact structure of optimal spanners. While minimizing the total weight of the spanner, considering Steiner points naturally allows for better results. Limited work is available on settings allowing for Steiner points [6].

We study the optimal geometric structure of solutions in a weight-restricted setting analytically. Specifically, we consider the *Restricted-Weight Minimum-Dilation Spanner Problem with Steiner Points* on geometric problem instances consisting of three points in $\mathbb{R}^2$. Given a maximum weight L, where weight is defined as the sum over the lengths of all edges, we thus want to find the spanner over $P \cup S$, for S a set of Steiner points that is optimal, that is the spanner minimizing the maximum dilation between any two points in P. Previous work on this problem by Kooijmans [13] and Verstege [18] has focused on providing algorithms for determining optimal spanners based on convex optimization.

We characterize the general topological structure of optimal spanners for all problem instances with $|P| = 3$.

*Department of Mathematics and Computer Science, Eindhoven University of Technology, k.a.buchin@tue.nl

†Institute of Computer Science, University of Bonn, haverkort@uni-bonn.de

For the case that the points in P are the vertices of an equilateral triangle, we provide a complete geometric description of the optimal spanner. Figure 1 contains examples of optimal spanners for this case. Likewise, we provide a description for the case of an isosceles triangle and low maximum weight L.

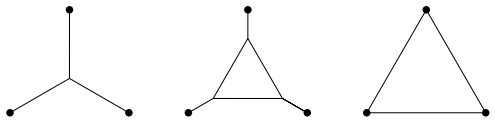


Figure 1: Optimal spanners for P consisting of the vertices of an equilateral triangle for increasing maximum weight

Analyzing geometric properties of optimal spanners serves the development of fast and well-performing approximation algorithms, by providing exploitable substructures and generally furthering the understanding of the problems at hand. As such, this work aims to aid further research into weight-restricted geometric spanners with Steiner vertices.

2 General Triangle

Let $P \subset \mathbb{R}^2, |P| = 3$, such that the set forms a triangle. Let these points be labeled A, B, C. As $G = (P \cup S, E)$ must be a connected graph, there must exist paths between each pair of points, which potentially overlap. Let $\alpha, \beta, \gamma : [0, 1] \to \mathbb{R}^2$ be curves describing the paths between A and B, B and C, and C and A respectively. Let furthermore $\alpha[x, y]$ denote the subcurve from $\alpha(x)$ to $\alpha(y)$ for $0 \leq x \leq y \leq 1$.

The proofs of Lemmas 1 to 5 are given in the appendix.

Lemma 1 *An optimal spanner has a maximal dilation of 1, or the total cost is maximal, thus* $\sum_{e \in E} |e| = L$.

Lemma 2 *In an optimal spanner, curves* α, β, γ *are injective.*

As a result of Lemma 2, we can denote $\alpha[x, y]$ by $\alpha[X \to Y]$ for $X = \alpha(x), Y = \alpha(y)$ with $0 \leq x, y \leq 1$. We furthermore denote the subcurve $\alpha[x, y]$ in reverse direction by $\alpha[Y \to X]$. We similarly denote the subcurves $\alpha[X \to Y]$ and $\alpha[Y \to X]$ excluding the endpoints by $\alpha(X \to Y)$ and $\alpha(Y \to X)$ respectively.

Lemma 3 *Let* X, Y *be points shared by curves* α, β *in an optimal spanner. Then*

$$\alpha[X \to Y] = \beta[X \to Y]$$

Similarly for pairs α, γ *and* β, γ.

Thus, between two points shared by two curves, the two curves consist of identical point sets.

Lemma 4 *In an optimal spanner for* $L \leq |AB| + |AC| + |BC|$*, curves* α, β, γ *consist of straight line segments between* A, B, C *and additional Steiner points.*

Lemma 5 *In an optimal spanner,* α, β, γ *are contained in the convex hull of* A, B, C.

Lemma 6 *An optimal spanner has the following form: a triangle with each vertex connected to exactly one of the points* A, B, C.

Proof. Let curve α be given arbitrarily. We then consider curves β, γ. As $\alpha(1) = \beta(0) = B$, and by Lemma 3, there must exist a point B' such that $\alpha[B \to B'] = \beta[B \to B'] = \alpha \cap \beta$. Analogously, there exist a point C' such that $\beta[C \to C'] = \gamma[C \to C'] = \beta \cap \gamma$ and a point A' such that $\alpha[A \to A'] = \gamma[A \to A'] = \alpha \cap \gamma$.

If $\alpha(A' \to B') \cap \beta \neq \emptyset$, let $X \in \alpha(A' \to B') \cap \beta$. By Lemma 2, $\alpha(A' \to B') \cap \beta(B \to B') = \emptyset$. Then by Lemma 3, $\alpha[X \to B']$ and $\beta[X \to B']$ must coincide. However, as in this case $\alpha[X \to B] = \beta[X \to B]$ and $B' \in \alpha(X \to B)$, this contradicts with the definition of B'. Therefore, $\alpha(A' \to B') \cap \beta = \emptyset$. Similarly, $\alpha(A' \to B') \cap \gamma = \emptyset$.

Thus in any optimal spanner, $\alpha(A' \to B') \cap \beta = \alpha(A' \to B') \cap \gamma = \emptyset$. Analogously, $\beta(B' \to C') \cap \alpha = \beta(B' \to C') \cap \gamma = \emptyset$ and $\gamma(A' \to C') \cap \alpha = \gamma(A' \to C') \cap \beta = \emptyset$.

We then, using Lemma 5, conclude that any optimal spanner must have the topology as shown in Figure 2. From Lemma 4 it follows that the curves between A and

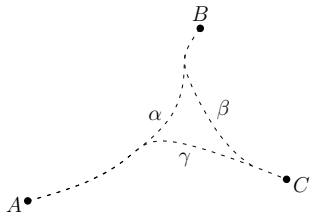


Figure 2: Optimal topology as proven in Lemma 6

A', B and B', C and C', A' and B', B' and C', A' and C', must be line segments. Thus the final form is as shown in Figure 3. □

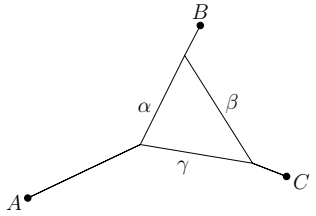


Figure 3: Optimal form as proven in Lemma 6

As used in the proof of Lemma 6, let A' be the point such that $\alpha[A \to A'] = \gamma[A \to A']$ and $|\alpha[A \to A']|$ maximal. Similarly, let B' be the point such that $\alpha[B \to B'] = \beta[B \to B']$ and $|\beta[B \to B']|$ maximal, and let C' be the point such that $\beta[C \to C'] = \gamma[C \to C']$ and $|\gamma[C \to C']|$ maximal.

Lemma 7 *In an optimal spanner, A', B', C' are distinct or $A' = B' = C'$.*

Proof. Assume that there exists an optimal spanner where this is not the case. Without loss of generality, assume that $A' = B'$, and thus $C' \neq A' = B'$. Then, $\beta[C, A'] = \gamma[C, A']$. Furthermore, by Lemma 6, and as A', C' are distinct, $|\gamma[C, A']| > |\gamma[C, C']|$. But this contradicts with the definition of C'. Thus, our assumption cannot hold. □

Lemma 8 *In an optimal spanner, $A' = B' = C'$ or $D_G(A, B) = D_G(A, C) = D_G(B, C)$.*

Proof. Assume $D_G(A, B) > D_G(B, C)$ in an optimal spanner. Then segment $B'C'$ can be moved parallelly inwards towards the center of triangle $A'B'C'$ to reduce total cost, while not altering the maximal dilation. Then, from Lemma 1, it follows that the spanner cannot be optimal. Thus, $D_G(A, B) \leq D_G(B, C)$ must hold for an optimal spanner with A', B', C' distinct. Similarly, because of the ability to move segment $A'B'$ parallelly inwards, $D_G(A, B) \geq D_G(B, C)$. By analogous arguments, for a spanner to be optimal with A', B', C' distinct, $D_G(A, B) = D_G(A, C) = D_G(B, C)$ must hold. □

The *dilation center* X^* of three points A, B, C is defined as the point X that minimizes the dilation of the graph with edges AX, BX and CX, where $P = A, B, C$. We call G^*, the graph with edges AX^*, BX^* and CX^*, the *minimum-dilation star*. The point X^* has been entered into the Encyclopedia of Triangle Centers as X(3513) [12]. It has been named the 1st Dilation Center, and was contributed by Eppstein [8, 9], who also observed that $D_{G^*}(A, B) = D_{G^*}(B, C) = D_{G^*}(A, C)$. Let L^* be the weight of G^*, that is, $L^* = |AX^*| + |BX^*| + |CX^*|$.

Lemma 9 *For any point M in the interior or on the boundary of ΔABC, $|AM| + |BM| + |CM| < |AB| + |AC| + |BC|$.*

See appendix for the proof of Lemma 9.

Lemma 10 *Let L_{MST} be the weight of the minimum Steiner tree over P. Then, for any weight L such that $L_{MST} \leq L \leq L^*$, any optimal spanner satisfies $A' = B' = C'$.*

Proof. We first show that for $L = L^*$, G^* is optimal. Assume that there exists another spanner G with cost at most L^*, and smaller maximum dilation. By its definition, G^* is optimal among spanners satisfying $A' = B' = C'$. Thus, in G, A', B', C' must be distinct. If, in G, X^* lies outside or of $\Delta A'B'C'$, then one of the curves α, β, γ, combined with the line segment connecting its endpoints, encloses X^*. W.l.o.g. assume this to be α. Therefore, $|\alpha| > |AX^*| + |BX^*|$, and $D_G(A, B) > D_{G*}(A, B)$. Thus, G would be suboptimal. Therefore X^* must lie on the boundary or in the interior of $\Delta A'B'C'$. But then, by Lemma 9, the cost of G equals $|AA'|+|BB'|+|CC'|+|A'B'|+|B'C'|+|A'C'| > |AA'| + |BB'| + |CC'| + |A'X^*| + |B'X^*| + |C'X^*| \geq |AX^*| + |BX^*| + |CX^*| = L^*$. Thus, an optimal spanner G cannot exist. Therefore, for $L = L^*$, the optimal spanner is the network with $A' = B' = C' = X^*$.

Now consider $L < L^*$. Suppose that there exists a weight $L < L^*$ for which an optimal spanner G exists with A', B', C' distinct. Then, by Lemma 8, $D_G = D_G(A, B) = D_G(A, C) = D_G(B, C)$. Since G^* is optimal, and G has weight at most $L < L^*$, we must have $D_G \geq D_{G^*}$. In G^*, the total length of curves α, β, γ equals $D_{G^*} \cdot (|AB|+|BC|+|AC|)$. Since each edge in G^* is contained in exactly two paths, this equals $2L^*$. However, in G, edges $A'B', A'C', B'C'$ are only contained in a single path. Thus, $D_G \cdot (|AB| + |BC| + |AC|) < 2L$. Then, $\frac{2L}{|AB|+|BC|+|AC|} > D_G \geq D_{G*} = \frac{2L^*}{|AB|+|BC|+|AC|}$. This contradicts with the definition of G as an optimal spanner with weight $L < L^*$. Therefore, $A' = B' = C'$ for any optimal spanner with weight $L \leq L^*$. □

Lemma 11 *For any weight L such that $L > L^*$, A', B', C' are distinct in any optimal spanner.*

Proof. By its definition, the dilation star is optimal among spanners satisfying $A' = B' = C'$. However, by Lemma 1, the dilation star cannot be optimal. As such, no spanner satisfying $A' = B' = C'$ is optimal. Then, by Lemma 7, A', B', C' must be distinct in any optimal spanner. □

3 Equilateral triangle

Lemma 12 *In an optimal spanner for an equilateral triangle, $d_G(A, B) = d_G(A, C) = d_G(B, C)$.*

Proof. By Lemma 7, we only need to consider the spanners in which A', B', C' all coincide or are all distinct. As by the definition of the equilateral triangle $|AB| = |AC| = |BC|$, it suffices to show that $D_G(A, B) = D_G(A, C) = D_G(B, C)$ for any optimal spanner.

The case where $A' = B' = C'$ holds then directly follows from Lemma 11 and the fact that the Fermat point and the dilation center coincide for equilateral triangles.

The case where A', B', C' are distinct directly follows from Lemma 8. □

Lemma 13 *Let for a triangle ΔABC the sum of the distances from the vertices to the Fermat point, $|a'| + |b'| + |c'|$, be given. Then the sum of the length of the triangle's edges, $|a| + |b| + |c|$, is minimal for ΔABC equilateral.*

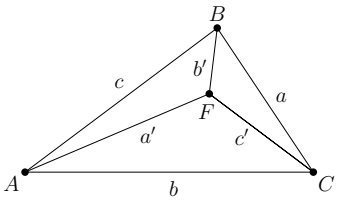


Figure 4: Illustration of notation used in the proof of Lemma 13

Proof. Suppose one of the angles of ΔABC is greater than or equal to 120° in an optimal spanner. Without loss of generality, assume $\angle ABC \geq 120°$ be given. Then, as the Fermat point of ΔABC coincides with point B, $|a'| + |b'| + |c'| = |a| + |c|$. By the law of sines:

$$\frac{|c|}{\sin(\angle ACB)} = \frac{|a|}{\sin(\angle BAC)} = \frac{|b|}{\sin(\angle ABC)} = d$$

where d is the diameter of the circumcircle. Then:

$$\begin{aligned} |b| &= d \cdot \sin(\angle ABC) \\ &= \frac{\sin(\angle ABC)(|a| + |c|)}{\sin(\angle BAC) + \sin(\angle ACB)} \end{aligned}$$

To find the minimum for $|a|+|b|+|c|$ given $|a'|+|b'|+|c'|$, we want to determine the minimum for $|b|$. As $\angle ABC$ and $|a|+|c|$ given, $\angle BAC$ suffices to describe the entire triangle. Thus:

$$\begin{aligned} \frac{d}{d\angle BAC}|b| &= \sin(\angle ABC) \cdot (|a| + |c|) \\ &\cdot \frac{\cos(180° - \angle BAC - \angle ABC) - \cos(\angle BAC)}{(\sin(\angle BAC) + \sin(180° - \angle BAC - \angle ABC))^2} \end{aligned}$$

Equating to zero and solving using the constraints that $120° \leq \angle ABC < 180°$ and that $\angle ABC + \angle ACB + \angle BAC = 180°$, we find an extremum at $\angle BAC = \angle ACB = \frac{180° - \angle ABC}{2}$. As the derivative increases for increasing $\angle BAC$, this must be a minimum. Thus, $|b| \geq \frac{\sqrt{3}}{2}(|a| + |c|) = \frac{\sqrt{3}}{2}(|a'| + |b'| + |c'|)$. Thus, $|a| + |b| + |c| \geq \frac{\sqrt{3}+2}{2}(|a'| + |b'| + |c'|)$

We compare this with ΔABC equilateral:

$$|a| + |b| + |c| = 3|a| = 3\sqrt{3}|a'| = \sqrt{3}(|a'| + |b'| + |c'|)$$

Thus, for $|a'| + |b'| + |c'|$ given, the sum of the length of the triangle's edges is smaller for an equilateral triangle than for a triangle with an angle greater than 120°.

Now assume $\angle BAC \leq 120°$, $\angle ABC \leq 120°$ and $\angle ACB \leq 120°$. Let $S = |a'| + |b'|$ be given. As the Fermat point F is also a Steiner point in the minimum Steiner tree of ΔABC, $\angle AFB = \angle BFC = \angle AFC = 120°$. Then, by the law of cosines:

$$\begin{aligned} |c|^2 &= |a'|^2 + |b'|^2 - 2|a'||b'|\cos(120°) \\ &= |a'|^2 + |b'|^2 + |a'||b'| \\ &= |a'|^2 + (S - |a'|)^2 + |a'|(S - |a'|) \\ &= |a'|^2 + S^2 - |a'|S \end{aligned}$$

As S is fixed, $|c|^2$ is solely dependent on $|a'|$. Therefore:

$$\frac{d|c|^2}{d|a'|} = 2|a'| - S$$

To find the minimum value for $|c|$ given S, we equate the derivative to zero.

$$\begin{aligned} & 2|a'| - S = 0 \\ \iff \quad & 2|a'| = S \\ \iff \quad & 2|a'| = |a'| + |b'| \\ \iff \quad & |a'| = |b'| \end{aligned}$$

Thus, $|c|$ is minimal for ΔABF isosceles. Then from simple trigonometry, utilizing $\angle AFB = 120°$, it follows that in this case, $|c| = \sqrt{3}|a'| = \sqrt{3}\frac{S}{2}$. Thus, more generally, $|c| \geq \sqrt{3}\frac{S}{2} = \sqrt{3}\frac{|a'|+|b'|}{2}$, where $|c| = \sqrt{3}\frac{|a'|+|b'|}{2}$ is only achieved for $|a'| = |b'|$.

Similarly for the other edges, by symmetry:

$$\begin{aligned} |a| &\geq \frac{\sqrt{3}}{2}(|b'| + |c'|) \\ |b| &\geq \frac{\sqrt{3}}{2}(|a'| + |c'|) \end{aligned}$$

Then, for the sum of the triangle's edges:

$$\begin{aligned} |a| + |b| + |c| \geq & \frac{\sqrt{3}}{2}(|b'| + |c'|) + \frac{\sqrt{3}}{2}(|a'| + |c'|) \\ & + \frac{\sqrt{3}}{2}(|a'| + |b') \\ = & \sqrt{3}(|a'| + |b'| + |c'|) \end{aligned}$$

As $|a| = \frac{\sqrt{3}}{2}(|b'| + |c'|) \iff |b'| = |c'|$, and as this also holds for $|b|, |c|$ by symmetry, $|a| + |b| + |c| = \sqrt{3}(|a'| + |b'| + |c'|) \iff |a'| = |b'| = |c'|$. Thus for $|a'| + |b'| + |c'|$ given, $|a|+|b|+|c|$ is minimal for ΔABC equilateral. □

Corollary 14 *For the sum of the edge lengths of a triangle ΔABC, $|a|+|b|+|c|$ given, the sum of the distances from the vertices to the Fermat point, $|a'| + |b'| + |c'|$ is maximal for ΔABC equilateral.*

Theorem 15 *Let ΔABC be an equilateral triangle. Then for a given $L \geq \sqrt{3} \cdot |AB|$, the maximal dilation is minimal for the following spanner: an equilateral triangle centered in ΔABC with its vertices connected to the vertices A, B, C such that its vertices lie on the line segments connecting A, B, C to the Fermat point of ΔABC.*

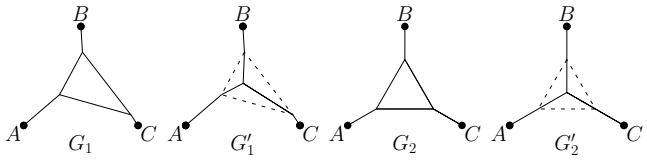


Figure 5: Graphs used in the proof of Theorem 15

Proof. W.l.o.g. assume $|AB| = 1$. Let G_1 be an optimal spanner which does not have this structure. By Lemma 6, it has the general form as depicted in Figure 5. Let s_1 be the sum of the edge lengths of its inner triangle, and t_1 the sum of the lengths of the segments connecting the vertices of the inner triangle and A, B, C. Let D_1 be its maximum dilation, $\max(D_{G_1}(A,B), D_{G_1}(A,C), D_{G_1}(B,C))$, and let L_1 be its total cost.

Then, $L_1 = s_1 + t_1$. Furthermore, as ΔABC equilateral, $|AB| = |AC| = |BC|$, and as by Lemma 12 $d_{G_1}(A,B) = d_{G_1}(A,C) = d_{G_1}(B,C)$, it follows that $D_{G_1}(A,B) = D_{G_1}(A,C) = D_{G_1}(B,C)$. Then:

$$\begin{aligned} 3D_1 &= 3 \cdot \max(D_{G_1}(A,B), D_{G_1}(A,C), D_{G_1}(B,C)) \\ &= D_{G_1}(A,B) + D_{G_1}(A,C) + D_{G_1}(B,C) \\ &= s_1 + 2t_1 \end{aligned}$$

Now consider a spanner G_2, with a centered equilateral triangle with its vertices on the line segments connecting A, B, C to the Fermat point of ΔABC, where the sum of the edge lengths of the inner triangle, s_2, equals s_1. Let t_2 be the sum of the distances between the inner triangle's vertices and A, B, C, L_2 its total cost and D_2 its maximum dilation. Then, as G_1 is an optimal spanner, $L_2 \geq L_1$ or $D_2 \geq D_1$. Next, we consider our previously found equations, utilizing $s_1 = s_2$:

$$\begin{aligned} & L_2 \geq L_1 \\ \iff \quad & s_2 + t_2 \geq s_1 + t_1 \\ \iff \quad & t_2 \geq t_1 \end{aligned}$$

$$\begin{aligned} & D_2 \geq D_1 \\ \iff \quad & 3D_2 \geq 3D_1 \\ \iff \quad & s_2 + 2t_2 \geq s_1 + 2t_1 \\ \iff \quad & t_2 \geq t_1 \end{aligned}$$

Thus, both statements are equivalent to $t_2 \geq t_1$.

Next consider replacing the edges of the inner triangles of G_1 and G_2 by three line segments between the vertices of the inner triangle to its Fermat point. Let these new spanners be G_1' and G_2'. Let u_1 and u_2 be the sum of the lengths of these three segments for G_1' and G_2' respectively. From the total cost of the minimal Steiner tree of ΔABC equaling $\sqrt{3} \cdot |AB|$, it follows that $t_1 + u_1, t_2 + u_2 \geq \sqrt{3} \cdot |AB|$.

First consider the case where the inner triangle of G_1 is equilateral. Then $u_1 = u_2$, but as it is not centered with its vertices on the line segments connecting A, B, C to the Fermat point of ΔABC, $t_1 + u_1 > \sqrt{3} \cdot |AB|$. By construction, G_2' is the minimum Steiner tree for ΔABC and thus $t_2 + u_2 = \sqrt{3} \cdot |AB|$. But then, as $u_1 = u_2$, $t_1 > t_2$.

Secondly, we consider the case where the inner triangle of G_1 is not equilateral. As G_2' is the minimum Steiner tree for ΔABC, $t_2 + u_2 = \sqrt{3} \cdot |AB|$. By Corollary 14 $u_1 < u_2$. But then, as $t_1 + u_1 \geq \sqrt{3} \cdot |AB|$, $t_1 > t_2$.

Thus in all cases $t_1 > t_2$. Thus G_1 cannot be optimal. □

Theorem 16 *Let points A, B, C, the vertices of an equilateral triangle, and $\sqrt{3} \cdot |AB| \leq L \leq 3 \cdot |AB|$, the maximum total cost, be given. Then, in an optimal spanner,*

$$\begin{aligned} A' &= A + \frac{\vec{AB} + \vec{AC}}{|\vec{AB} + \vec{AC}|} \cdot \frac{3 \cdot l_o - L}{3\sqrt{3} - 3}, \\ B' &= B + \frac{\vec{BA} + \vec{BC}}{|\vec{BA} + \vec{BC}|} \cdot \frac{3 \cdot l_o - L}{3\sqrt{3} - 3}, \\ C' &= C + \frac{\vec{CA} + \vec{CB}}{|\vec{CA} + \vec{CB}|} \cdot \frac{3 \cdot l_o - L}{3\sqrt{3} - 3} \end{aligned}$$

with $l_o = |AB|$.

See appendix for the proof of Theorem 16.

4 Isosceles triangle

Let ΔABC be an isosceles triangle with $|AB| = |BC|$. We analyze the case of small L, that is for $L \leq L^*$, for which we know by Lemma 10 that $A' = B' = C'$.

Lemma 17 *In a spanner with $X := A' = B' = C'$, for X at a given height with respect to base AC, $|AX| + |CX|$ is minimal for $|AX| = |CX|$.*

See appendix for the proof of Lemma 17.

Corollary 18 *In an optimal spanner, if $X := A' = B' = C'$, this point lies on the perpendicular bisector of AC.*

Proof. Let G be a given optimal spanner, in which $|AX| \neq |CX|$. Without loss of generality due to the symmetry of an isosceles triangle, assume $D_G(A,B) \geq D_G(B,C)$. We will consider a spanner G^*, in which $|AX^*| = |CX^*|$, but where $|X^*X'^*|$ equals $|XX'|$ in G, where X' is the perpendicular projection of X on AC. Then, by applying Lemma 17 and the Pythagorean Theorem to $|BX|$, it follows that the total cost of G^* is smaller than the total cost of G.

We further show that the maximal dilation is reduced. As $|AX| + |CX|$ minimal for $|AX| = |CX|$, $D_G(A,C) > D_{G^*}(A,C)$. By the Pythagorean Theorem, $|AX^*| < |AX|$ and $|BX^*| < |BX|$. But then, $D_G(A,B) > D_{G^*}(A,B) = D_{G^*}(B,C)$. But then, G cannot be optimal, and thus for any optimal spanner, $|AX| = |CX|$ must hold. □

Together with Lemma 1 and Lemma 9 this provides a complete characterization of the spanner for the case that $L_{MST} \leq L \leq L^*$, where L_{MST} is the weight of the minimum Steiner tree and L^* is the weight of the minimum dilation star. We have not yet found similar characterizations for larger L. Conjectures based on computational experiments are included in Section 5.

5 Computational experimentation

To further investigate properties of optimal spanners, we used an approximation algorithm to find such spanners for various problem instances. We chose not to use exact algorithms like those developed by Kooijmans [13] and Verstege [18] due to the simplicity of the considered cases. Consequently, simple approximation algorithms, such as the evolution-based approach we implemented, give satisfactory results while avoiding error-prone complex implementations.

In Figure 6, for several problem instances and for increasing maximum weight, the computationally found approximations are visualized. From the development of the spanner for increasing maximum weight, we deduce multiple conjectures.

Conjecture 1 *Let ΔABC be an isosceles triangle with $|AB| = |BC|$. Then the optimal spanner is symmetric in the perpendicular bisector of AC.*

Conjecture 2 *Let ΔABC be an isosceles triangle with $|AB| = |BC|$. If $\angle ABC \leq 120°$, $|AA'| > 0, |BB'| > 0, |CC'| > 0$ in the optimal spanner for $|AF| + |BF| + |CF| < L < |AB| + |AC| + |BC|$ with F the Fermat point of ΔABC.*

Note that in Conjecture 2, the lower and upper bounds on L are given by the cost of the minimum Steiner tree and the complete graph respectively.

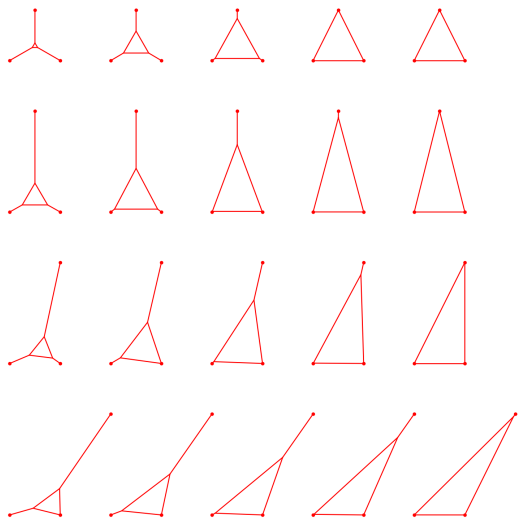


Figure 6: Approximately optimal spanners for various problem instances of three points at varying maximum weight

Conjecture 3 *Let L^* be the weight of the dilation star. Then $\lim_{L \downarrow L^*} \angle A'B'C' = \lim_{L \downarrow L^*} \angle A'C'B' = \lim_{L \downarrow L^*} \angle B'A'C' = 60°$.*

Conjecture 4 *If in an optimal spanner $|AA'|, |BB'|, |CC'| > 0$, the lines AA', BB', CC' intersect in a single point.*

6 Conclusion

The aim of our work was to analyze the geometry of solutions to the Restricted-Weight Minimum-Dilation Spanner Problem with Steiner Points. As presented, we have determined the topology of solutions for instances with $|P| = 3$. Additionally, we can fully describe the optimal spanner if the points in P are the vertices of an equilateral triangle, and partially (i.e. if the maximum weight is small) if they form an isosceles triangle.

We leave multiple open problems. Proofs for optimal spanners for isosceles triangles and larger weight are still elusive. Similarly, descriptions of optimal spanners and corresponding proofs for general triangles, quadrilaterals (in particular the square), pentagons and further polygons are yet to be found. In restricted settings, the previous work by Kooijmans [13] and Verstege [18] provides insights into the topologies of solutions. Considering optimal spanners more generally, related problems include solutions and their structure for other objective functions in weight-restricted settings, for example, minimizing the diameter or the number of edges.

References

[1] R. Ahmed, K. Hamm, M. J. Latifi Jebelli, S. Kobourov, F. Darabi Sahneh, and R. Spence. Approximation algorithms and an integer program for multi-level graph spanners. *CoRR*, abs/1904.01135, 2019.

[2] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[3] S. Arya, D. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 703–712, 1994.

[4] S. Arya, D. M. Mount, and M. Smid. Dynamic algorithms for geometric spanners of small diameter: Randomized solutions. *Computational Geometry*, 13(2):91–107, 1999.

[5] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17(1):33–54, 1997.

[6] G. Borradaile and D. Eppstein. Near-linear-time deterministic plane Steiner spanners forwell-spaced point sets. *Computational Geometry*, 49:8–16, 2015.

[7] P. Carmi and L. Chaitman-Yerushalmi. Minimum weight Euclidean t-spanner is NP-hard. *Journal of Discrete Algorithms*, 22:30–42, 2013.

[8] D. Eppstein. *The dilation center of a triangle.* https://11011110.github.io/blog/2008/07/10/dilation-center-of.html, 2008.

[9] D. Eppstein and K. A. Wortman. Minimum dilation stars. *Computational Geometry*, 37(1):27–37, 2007.

[10] A. M. Farley, A. Proskurowski, D. Zappala, and K. Windisch. Spanners and message distribution in networks. *Discrete Applied Mathematics*, 137(2):159–171, 2004.

[11] J. Gudmundsson, G. Narasimhan, and M. Smid. Applications of geometric spanner networks. In *Encyclopedia of Algorithms*, pages 40–43. Springer, 2008.

[12] C. Kimberling. *Encyclopedia of Triangle Centers.* https://faculty.evansville.edu/ck6/encyclopedia/ETCPart3.html, 2020.

[13] P. J. A. M. Kooijmans. *Fixed-weight minimum-dilation networks.* Master's thesis, Eindhoven University of Technology, Department of Mathematics and Computer Science, 2008.

[14] H. Le and S. Solomon. Truly optimal Euclidean spanners. *CoRR*, abs/1904.12042, 2019.

[15] G. Narasimhan and M. Smid. *Geometric Spanner Networks.* Cambridge University Press, 2007.

[16] G. Navarro and R. Paredes. Practical construction of metric t-spanners. In *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments*, pages 69–81, 2003.

[17] M. Sigurd and M. Zachariasen. Construction of minimum-weight spanners. In *Proceedings 12th European Symposium on Algorithms*, pages 797–808, 2004.

[18] M. Verstege. *Fixed-weight minimum-dilation networks.* Master's thesis, Eindhoven University of Technology, Department of Mathematics and Computer Science, 2011.

Appendix

Proof for Lemma 1

Proof. Assume there exists an optimal spanner where the maximal dilation is greater than 1, and the total cost is not maximal. Let $L_u < L$ be the total weight of the optimal spanner.

Firstly, we will consider the case that a single path has maximal dilation. W.l.o.g. assume that α is said path. Thus, $D_G(A,B) > D_G(B,C)$, $D_G(A,B) > D_G(A,C)$. Then, as $D_G(A,B) > 1$, α is not a single line segment. Then we can use the available weight $L - L_u$ to decrease $d_G(A,B)$, thereby decreasing $D_G(A,B)$. Thus the solution is not optimal.

Next, we consider the case that two paths have maximal dilation. W.l.o.g. assume that α, β are said paths. Thus, $D_G(A,B) = D_G(B,C) > D_G(A,C)$. Then, as $D_G(A,B) = D_G(B,C) > 1$, neither α nor β is a single line segment. Then we can use half of the available weight $L - L_u$ to decrease $d_G(A,B)$ and half to decrease $d_G(B,C)$, thereby decreasing $D_G(A,B), D_G(B,C)$. Thus the solution is not optimal.

Finally, we consider the case where $D_G(A,B) = D_G(B,C) = D_G(A,C)$. Then, as $D_G(A,B) = D_G(B,C) = D_G(A,C) > 1$, none of the paths consist of a single line segments. Thus we can use a third of the available weight $L - L_u$ for decreasing $d_G(A,B), d_G(B,C), d_G(A,C)$ each respectively, thereby decreasing the maximal dilation. Thus the solution is not optimal.

Thus in all cases, the presumed solution is not optimal. □

Proof for Lemma 2

Proof. W.l.o.g. we will consider α. Let α not be injective. Then there thus exist $0 \leq x < y \leq 1$ such that $\alpha(x) = \alpha(y)$.

If $\alpha[x,y] \setminus (\beta \cup \gamma) \neq \emptyset$, redefining α to the concatenation of $\alpha[0,x]$ and $\alpha[y,1]$, will decrease the total cost while not increasing the maximum dilation. Thus by Lemma 1, the solution cannot be optimal.

Otherwise, $(\beta \cup \gamma) \cap \alpha[x,y] = \alpha[x,y]$. Then let $b_j = \inf\{b : \beta(b) \in \alpha[x,y]\}, c_j = \inf\{c : \gamma(c) \in \alpha[x,y]\}$ be the indices at which β, γ join $\alpha[x,y]$ respectively, and let $b_l = \sup\{b : \beta(b) \in \alpha[x,y]\}, c_l = \sup\{c : \gamma(c) \in \alpha[x,y]\}$ be the indices at which they leave $\alpha[x,y]$. If $|\beta[b_j,b_l]| > \frac{|\alpha[x,y]|}{2}$, we can redefine $\beta[b_j,b_l]$ to $\alpha[x,y] \setminus \beta[b_j,b_l]$, resulting in $|\beta[b_j,b_l]| \leq \frac{|\alpha[x,y]|}{2}$ without increasing dilation. Analogously, we can redefine $\gamma[c_j,c_l]$ such that $|\gamma[c_j,c_l]| \leq \frac{|\alpha[x,y]|}{2}$. Then, if $\alpha[x,y] \setminus (\beta \cup \gamma) \neq \emptyset$ for the redefined curves, the modified solution, and therefore the original solution, cannot be optimal as previously described. Otherwise, $|\beta[b_j,b_l]| = |\gamma[c_j,c_l]| = \frac{|\alpha[x,y]|}{2}$, and we can redefine $\beta[b_j,b_l]$ to $\gamma[c_j,c_l]$ without increasing dilation. Then $\alpha[x,y] \setminus (\beta \cup \gamma) \neq \emptyset$, and thus the modified solution, and therefore the original solution cannot be optimal as previously shown.

Thus, in all cases the solution is not optimal, and we conclude that α must be injective in an optimal solution. □

Proof for Lemma 3

Proof. Let X, Y be points shared by curves α, β. Assume that $\alpha[X \to Y] \neq \beta[X,Y]$ in an optimal solution. Then $\alpha[X,Y] \setminus \beta[X,Y] \neq \emptyset$ and $\beta[X,Y] \setminus \alpha[X,Y] \neq \emptyset$.

Suppose that $|\alpha[X,Y] \setminus \beta[X,Y]| < |\beta[X,Y] \setminus \alpha[X,Y]|$. Then we can redefine $\beta[X,Y]$ to $\alpha[X,Y]$ without increasing dilation, while maintaining or lowering the total cost. The cost is only maintained if $\beta[X,Y] \setminus \alpha[X,Y] \subseteq \gamma$. Then, $\gamma \cap (\beta[X,Y] \setminus \alpha[X,Y])$ can also be redefined to segments of $\alpha[X,Y]$ without increasing the dilation, while reducing the cost. By Lemma 1, the modified solution, and therefore the original solution, cannot be optimal. Thus, $|\alpha[X,Y] \setminus \beta[X,Y]| \geq |\beta[X,Y] \setminus \alpha[X,Y]|$. Analogously, $|\alpha[X,Y] \setminus \beta[X,Y]| \leq |\beta[X,Y] \setminus \alpha[X,Y]|$.

Thus, $|\alpha[X,Y] \setminus \beta[X,Y]| = |\beta[X,Y] \setminus \alpha[X,Y]|$. Then, as previously shown, we can redefine $\beta[X,Y] \setminus \alpha[X,Y]$, and possibly $\gamma \cap (\beta[X,Y] \setminus \alpha[X,Y])$, to $\alpha[X,Y]$ to reduce the total cost without increasing the dilation. As such, by Lemma 1, the modified solution, and therefore the original solution, cannot be optimal.

Analogously for pairs α, γ and β, γ. □

Proof of Lemma 4

Proof. If $L = |AB| + |AC| + |BC|$, the optimal solution is given by the complete graph, which consists of straight line segments.

If $L < |AB| + |AC| + |BC|$, w.l.o.g. let us consider α. Let α contain a non-straight arc in an optimal solution. Let $\alpha[x,y]$ be such an arc, with $0 < x < y < 1$.

If $\alpha[x,y] \cap \beta = \alpha[x,y] \cap \gamma = \emptyset$, replacing $\alpha[x,y]$ with a direct line segment decreases cost, while not increasing dilation. By Lemma 1, the modified solution, and therefore the original solution cannot be optimal.

If $\alpha[x,y] \cap \beta \cap \gamma = \alpha[x,y]$, α, β, γ can all be redefined to a direct line segment to decrease cost, while not increasing dilation. By Lemma 1, the modified solution, and therefore the original solution cannot be optimal.

If $\alpha[x,y] \cap \beta = \alpha[x,y]$ and $\alpha[x,y] \cap \gamma = \emptyset$, both α, γ can be redefined to a direct line segment to decrease the cost, while not increasing dilation. By Lemma 1, the modified solution, and therefore the original solution cannot be optimal. By similar argument, the same holds for $\alpha[x,y] \cap \beta = \emptyset$ and $\alpha[x,y] \cap \gamma = \alpha[x,y]$.

Otherwise, we can introduce $z_1 < z_2 < \ldots < z_n$, with $z_1 > x, z_n < y$, such that $\alpha[x,z_1], \alpha[z_1,z_2], \alpha[z_2,z_3], \ldots, \alpha[z_{n-1},z_n], \alpha[z_n,y]$ all either reflect one of the previous cases or are a straight line segment.

Thus, in all cases, non-straight arcs result in a non-optimal solution. □

Proof for Lemma 5

Proof. By Lemma 4, we can consider the solution to be a network over a set of points V, consisting of the triangle's vertices and additional Steiner points.

Let $h_a(p)$ for $p \in V$ denote the distance from p to the line BC if the line segment connecting p and A intersects the

line BC, and the additive inverse of the distance from p to the line BC otherwise.

Assume that there exists a point q such that $h_a(q)$ positive in an optimal solution. Then let $q_1, q_2, \ldots, q_n \in V$ denote the points with $h_a(q_1) = h_a(q_2) = \ldots = h_a(q_n) = \max_{p \in V} h_a(p)$. Now move all q_i for $1 \leq i \leq n$ perpendicular to the line BC by ϵ towards said line, for ϵ sufficiently small. Then any line segment between q_i, q_j for $1 \leq i < j \leq n$, is not changed in length. For segments between p', q', where $h_a(p') < h_a(q') = \max_{p \in V} h_a(p)$, by the Pythagorean Theorem, the length is reduced. All remaining segments are not affected.

As $h_a(q) > 0$, and as each point in V is (in)directly connected to A, B, C, there must exist p', q' such that $h_a(p') < h_a(q') = \max_{p \in V} h_a(p)$ and p', q' directly connected. Thus, there must exist a segment which has been shortened, while none of the segments has increased in length. As such, the modified solution has a lower cost. By Lemma 1, the modified solution can thus not be optimal. Furthermore, as the maximum dilation has thus also not increased, the original solution can also not be optimal. As such, our assumption that there exists a point q such that $h_a(q) > 0$ cannot hold.

Analogously, an optimal solution cannot contain a point q such that the line segment connecting q and B intersects the line AC, or such that the line segment connecting q and C intersects the line AB. □

Proof for Lemma 9

Proof. For M in the interior of ΔABC, we trivially have $|AM| + |MB| < |AC| + |CB|$. Similarly, $|BM| + |MC| < |BA| + |AC|$ and $|AM| + |MC| < |AB| + |BC|$. Adding these equations gives $2 \cdot (|AM| + |BM| + |CM|) < 2 \cdot (|AB| + |BC| + |AC|)$.

For M on the boundary of ΔABC, $|AM| + |MB| \geq |AC| + |CB|$ if and only if $M = C$. Similarly, $|BM| + |MC| \geq |BA| + |AC|$ if and only if $M = A$ and $|AM| + |MC| \geq |AB| + |BC|$ if and only if $M = B$. Thus, as A, B, C are distinct, for any point on the boundary, $2 \cdot (|AM| + |BM| + |CM|) < 2 \cdot (|AB| + |BC| + |AC|)$ must hold. □

Proof for Theorem 16

Proof. Let points A, B, C, the vertices of an equilateral triangle, and L, the maximum total cost, be given. We will now construct A', B', C'. Let $l_o := |AB| = |AC| = |BC|$. Furthermore, let $l_c := |AA'|$, and let $l_i := |A'B'|$. Then, by Theorem 15, $l_c = |AA'| = |BB'| = |CC'|$ and $l_i = |A'B'| = |A'C'| = |B'C'|$.

Then, as by Theorem 15 $\angle A'AC = 30°$, $l_o = 2 \cdot l_c \cdot \cos(30°) + l_i$. From this equation, it follows that $l_i = l_o - 2 \cdot l_c \cdot \cos(30°)$. Furthermore, by Lemma 1, $3 \cdot l_c + 3 \cdot l_i = L$. Then we can solve for l_c:

$$\begin{aligned}
& 3 \cdot l_c + 3 \cdot l_i = L \\
\iff \quad & 3 \cdot l_o - 6 \cdot l_c \cdot \cos(30°) + 3 \cdot l_c = L \\
\iff \quad & 3 \cdot l_o - 3\sqrt{3} \cdot l_c + 3 \cdot l_c = L \\
\iff \quad & (3 - 3\sqrt{3}) \cdot l_c = L - 3 \cdot l_o \\
\iff \quad & l_c = \frac{3 \cdot l_o - L}{3\sqrt{3} - 3}
\end{aligned}$$

Finally, as by Theorem 15 $\angle A'AB = \angle A'AC$ and $|AB| = |AC|$:

$$A' = A + \frac{\vec{AB} + \vec{AC}}{|\vec{AB} + \vec{AC}|} \cdot \frac{3 \cdot l_o - L}{3\sqrt{3} - 3}$$

And by symmetry analogously for B' and C'. □

Proof for Lemma 17

Proof. Define X' as the perpendicular projection of X on AC, such that $\angle AX'X = 90°$, which is in any case valid by Lemma 5. Now let $|XX'|$ be fixed. Then:

$$\begin{aligned}
|AX| + |CX| =& \sqrt{|AX'|^2 + |XX'|^2} + \sqrt{|CX'|^2 + |XX'|^2} \\
=& \sqrt{|AX'|^2 + |XX'|^2} \\
& + \sqrt{(|AC| - |AX'|)^2 + |XX'|^2}
\end{aligned}$$

To find the minimum, we differentiate with respect to $|AX'|$:

$$\begin{aligned}
\frac{d(|AX| + |CX|)}{d|AX'|} =& \frac{|AX'|}{\sqrt{|AX'|^2 + |XX'|^2}} \\
& - \frac{|AC| - |AX'|}{\sqrt{(|AC| - |AX'|)^2 + |XX'|^2}} \\
=& \frac{|AX'|}{\sqrt{|AX'|^2 + |XX'|^2}} \\
& - \frac{|CX'|}{\sqrt{|CX'|^2 + |XX'|^2}}
\end{aligned}$$

Setting the derivative equal to zero, we find:

$$\begin{aligned}
& \frac{d(|AX| + |CX|)}{d|AX'|} = 0 \\
\iff \quad & \frac{|AX'|}{\sqrt{|AX'|^2 + |XX'|^2}} = \frac{|CX'|}{\sqrt{|CX'|^2 + |XX'|^2}} \\
\iff \quad & |AX'|^2(|CX'|^2 + |XX'|^2) = |CX'|^2(|AX'|^2 + |XX'|^2) \\
\iff \quad & |AX'|^2 |XX'|^2 = |CX'|^2 |XX'|^2 \\
\iff \quad & |AX'|^2 = |CX'|^2
\end{aligned}$$

And as lengths of line segments are non-negative, $|AX'| = |CX'|$. But then, by the Pythagorean Theorem, $|AX| = |CX|$. □

Fréchet Distance Between Two Point Sets

Maike Buchin* Bernhard Kilgus†

Abstract

We define and study the (discrete) Fréchet distance between two point sets in the plane. One variant of the well-known Fréchet distance seeks to find a polygonal curve on a point set with small (discrete) Fréchet distance to another given polygonal curve. Here, we consider two given point sets and ask if permutations of theses point sets exist, such that the (discrete) Fréchet distance of curves defined by the permutations is small.

1 Introduction

The (discrete) Fréchet distance is a popular measure between two polygonal curves that takes the order of the points along the curves into account. Intuitively, the similarity of the shapes of the two curves is measured. In this paper, we want to measure the similarity between two point sets by asking if we can find an order of the points, such that the (discrete) Fréchet distance between the polygonal curves on the point sets in this order is small. That is, we ask if the point sets can be ordered such that the resulting curves have a similar shape.

This is a natural problem within the Fréchet distance research that has not been considered yet, and there are some interesting applications of this approach. For instance, it can be used to detect similar linear formations between two groups where one group is represented by one point for each entity within the group. Furthermore, for two sets of GPS-positions of two entities for which the time component is missing, we want to determine if the positions can be ordered in such a way that the entities were moving along a similar route.

1.1 Problem Definition

A polygonal curve is a curve $P\colon [1,n] \to \mathbb{R}^2$ where n is a positive integer, such that the restriction of P to the interval $[i, i+1]$ is affine, that is $P(i+\lambda) = (1-\lambda)P(i) + \lambda P(i+1)$, for each $i \in \{1, 2, \ldots, n-1\}$. We call $\{P(i) \mid i \in [1:n]\}$ the vertex set of P. To measure the dissimilarity between two polygonal curves, we use the Fréchet distance [6] and the discrete Fréchet distance [11]. For two polygonal curves $P, Q\colon [1,n] \to \mathbb{R}^2$ and $Q\colon [1,m] \to \mathbb{R}^2$ their *Fréchet distance* is defined as

$$\delta_F(P,Q) = \inf_{\substack{\alpha\colon [0,1]\to[1,n] \\ \beta\colon [0,1]\to[1,m]}} \max_{t\in[0,1]} ||P(\alpha(t)) - Q(\beta(t))||,$$

where α (resp. β) ranges over all continuous non-decreasing onto mappings from $[0,1]$ to $[1,n]$ (resp. $[1,m]$).

The discrete Fréchet distance is typically defined by *couplings*. Let $u(P)$ be the vertex set of P and $v(Q)$ the vertex set of Q. A coupling L between P and Q is a sequence $(u_{a_1}, v_{b_1}), (u_{a_2}, v_{b_2}), \ldots, (u_{a_j}, v_{b_j})$ of distinct pairs from $u(P) \times v(Q)$ such that $a_1 = 1$, $b_1 = 1$, $a_j = n$, $b_j = m$, and for all $i \in \{1, 2, \ldots, j\}$ we have $a_{i+1} = a_i$ or $a_{i+1} = a_i + 1$ and $b_{i+1} = b_i$ or $b_{i+1} = b_i + 1$. The length of L is defined as $\max_i d(u_{a_i}, v_{b_i})$, where d denotes the Euclidean distance between two points. The discrete Fréchet distance between P and Q equals the shortest possible length of a coupling between P and Q.

Now we define the (discrete) Fréchet distance between point sets. For this, we use the same notation, $\delta_{dF}(S,R)$ and $\delta_F(S,R)$, as for the (discrete) Fréchet distance between two polygonal curves.

Definition 1 *Given a point set $S \subset \mathbb{R}^2$ of size $|S| = n$. A polygonal curve $c(S)$* induced *by the point set S is a polygonal curve with vertex set S. That is $c(S)$ visits each point of S exactly once.*

Definition 2 *Let $S, R \subset \mathbb{R}^2$ be two point sets of size $|S| = n$, $|R| = m$. We define the (discrete) Fréchet distance between S and R as the smallest possible (discrete) Fréchet distance between a polygonal curve $c(S)$ induced by S and a polygonal curve $c(R)$ induced by R.*

Next we formally state the problem that we study.

Problem 1 *Given two point sets we ask if the (discrete) Fréchet distance between the point sets is at most a given value $\varepsilon > 0$ or not. If so, we want to find two induced curves with (discrete) Fréchet distance at most ε.*

1.2 Related Work

Alt and Godau [6] gave algorithms for computing the Fréchet distance between polygonal curves. For two polygonal curves P and Q of complexity n and m their algorithm decides in $\mathcal{O}(nm)$ time whether the Fréchet distance between P and Q is smaller than a given value $\varepsilon > 0$, and computes the exact Fréchet distance in

*Department of Mathematics, Ruhr University Bochum, Bochum, Germany, Maike.Buchin@rub.de

†Department of Mathematics, Ruhr University Bochum, Bochum, Germany, Bernhard.Kilgus@rub.de

$\mathcal{O}(nm\log(nm))$ time [6]. The discrete Fréchet distance was introduced by Eiter and Mannila, who gave an algorithm to compute it for two polygonal curves in $\mathcal{O}(nm)$ time [11]. Recently, Bringmann gave conditional quadratic lower bounds [9] for the Fréchet distance and discrete Fréchet distance. At nearly the same time, slightly faster algorithms were given for both the discrete Fréchet distance [4] and the Fréchet distance [10].

Eiter and Mannila defined a related distance measure, the link distance, between two points sets as the mimimum sum of point to point distances of a relation $\mathcal{R} \subset S_1 \times S_2$ for point sets S_1 and S_2. For each point s_1 of S_1, $\mathcal{R}$ must contain a pair (s_1, s_2) and vice versa [12].

To the best of our knowledge, the problem of finding polygonal curves on two given point sets with small Fréchet distance has not been considered previously. However, the problem where one polygonal curve P is given and the objective is to find a polygonal curve Q on one point set S with small (discrete) Fréchet distance to P is well studied [1, 13, 16]. There are several variants of this problem. The *all-points* variant requires all points of S to be visited by Q, whereas the *subset* variant allows to build the polygonal curve on a subset of S. Furthermore, one can allow that points of S are visited more than once (*non-unique*) or not (*unique*).

For the discrete Fréchet distance, Wylie and Zhu [16] showed that both non-unique variants are solvable in $\mathcal{O}(nm)$ time, where n is the complexity of P and m denotes the size of S. However, they showed that both unique variants are NP-complete. For the continuous Fréchet distance, Accisano and Üngör [1] showed NP-completeness for both unique variants and for the all-points, non-unique variant. Maheshwari et al. [13] showed that the continuous subset non-unique variant is solvable in $\mathcal{O}(nm^2)$ time. Assisano and Üngör introduced two extensions of the problem, namely allowing an affine transformation of P to be close to some curve on the point set S under the Fréchet distance [2] and dealing with imprecise point sets [3]. However, as for most of the techniques the order of the given curve is inherent, these approaches do not carry to our problem.

1.3 Contribution

We consider the discrete and continuous Fréchet distance between two point sets. For a curve and a point set, the discrete variant is NP-hard. However, we first observe that for two given point sets, the discrete Fréchet distance is equivalent to the Hausdorff distance. Then we show how to obtain two curves with minimal discrete Fréchet distance in $\mathcal{O}((m+n)\log(mn))$ time.

In contrast to the discrete variant, we prove that the continuous variant is NP-hard to approximate for any constant factor. We develop an exponential time algorithm to decide whether $\delta_F(S, R) \leq \varepsilon$ for two point sets S and R and a value $\varepsilon > 0$ that exploits the given geometry in order to reduce the number of possible permutations of the points significantly.

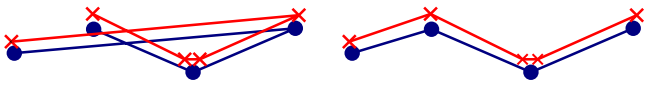


Figure 1: Induced curves with discrete Fréchet distance $< \varepsilon$ with and without crossings.

2 Discrete Fréchet Distance Between Point Sets

In this section, we consider the discrete Fréchet distance between two point sets and show how to obtain two induced curves that respect this distance.

We first compare the discrete Fréchet distance with the Hausdorff distance between two point set. Recall that the Hausdorff distance between two point sets S and R is defined as the smallest value ε such that for every point $s \in S$ there exists a point $r \in R$ with distance at most ε to s, and such that for every point $r \in R$ there exists a point s in S with distance at most ε to r. Hence, we observe, that for two point sets $S, R \subset \mathbb{R}^2$, $\delta_{dF}(S, R) = \delta_H(S, R)$, where $\delta_H(S, R)$ denotes the Hausdorff distance. The Hausdorff distance and thus the discrete Fréchet distance can be computed in $O(m+n)\log(m+n)$ time using Voronoi Diagrams as shwon by Alt et al. [5].

In addition to computing the value of the discrete Fréchet distance, we also want to find induced curves realizing the distance. First observe, that there will typically be many curves realizing a certain distance, see Figure 1. We show how to obtain curves with discrete Fréchet distance equal to the discrete Fréchet distance of the point sets that have a "nice", i.e., relatively short and non self-intersecting, representative curve. That is, we reveal a simple shape that implies an order of the point sets such that the discrete Fréchet distance between induced curves respecting this order is small.

To find such curves on S and R, we first determine a coupling between S and R. For this, we look for a partition of the points sets S and R into a sequence $\mathcal{M}$ of pairs of nonempty subsets (S_i, R_i) such that $\bigcup_i S_i = S$, $\bigcup_i R_i = R$, $S_i \cap S_j = \emptyset$, $R_i \cap R_j = \emptyset$ for all $i \neq j$, $|S_i| = 1$ or $|R_i| = 1$ for all i, and for all i, $d(s, r) \leq \varepsilon$ for all $s \in S_i$, $r \in R_i$. Thus, $\mathcal{M}$ defines a coupling on the points of S and R that realizes $\delta_{dF}(c(S), c(R)) \leq \varepsilon$. The curve $c(S)$ ($c(R)$) is obtained by connecting the points of S_i (R_i) arbitrarily and by connecting these subpaths in the order given by $\mathcal{M}$.

To compute $\mathcal{M}$, we build a dictionary D such that $\mathcal{M}$ consists of the key-value pairs of D, where a key is a singleton entry of a coupling pair (S_i, R_i). We assume that the coordinates of all points in S and R are unique.

In a first step, we map each point $s \in S$ to its closest point $r \in R$ and add an entry with key r and an assigned

list $\{s\}$ to D, or append the point s to the list if an entry with key r already exists in D. During this step, some points are left in R with no entry in D. Thus, in a second step, we map each of these points to its closest point in S, and update the entries of D in order to maintain the properties of $\mathcal{M}$. The following pseudocode describes this approach in more detail. $CP(s)$ denotes the closest point of $s \in S$ in R and $\hat{R}$ is a copy of R.

```
foreach s ∈ S do
    r = CP(s)
    if r ∈ D then D[r].add(s);
    else D.insert({r : {s}});
    R̂.remove(r)
foreach r ∈ R̂ do
    s = CP(r)
    if s ∈ D then D[s].add(r) ;
    else
        r' = CP(s)
        if len(D[r']) > 1 then D[r'].remove(s)
            D.insert({s : {r}}) ;
        else D.remove(r')
            D.insert({s : {r, r'}}) ;
i = 1
foreach s ∈ S do
    if s ∈ D then S_i = {s}, R_i = D[s], i++;
foreach r ∈ R do
    if r ∈ D then S_i = D[r], R_i = {r}, i++;
```

Algorithm 1: Construction of the partition $\mathcal{M}$

It is easy to verify, that the partition $\mathcal{M}$ obtained by Algorithm 1 has the demanded properties. Note also, that the singleton in each coupling pair is the closest point of one of its coupled points. Using the Voronoi Diagrams of S and R, computing $CP(s \in S)$ ($CP(r \in R)$) takes $\mathcal{O}(\log(m)$ ($\mathcal{O}(\log(n))$ time and hence the first two loops can be performed in $\mathcal{O}(n\log(m)+m\log(n))$ time. All operations on D take constant time using hashing. The Voronoi diagrams of S and R can be computed $O(m\log(m)+n\log(n))$ time. Thus we obtain:

Theorem 1 *Given two point sets $S, R \subset \mathbb{R}^2$, where $|S| = n$ and $|R| = m$, we can compute induced polygonal curves $c(S)$ and $c(R)$, such that $\delta_{dF}(c(S), c(R)) = \delta_{dF}(S, R)$ in $\mathcal{O}((m+n)\log(mn))$ time.*

Next, we describe how to obtain a representative curve c_{rep} with the preferred properties, i.e. simple and short, and induced curves with small discrete Fréchet distance to c_{rep}. Note that a curve $c(T)$ of shortest length induced by a point set T is also crossing free, since untangling a crossing only gives a shorter curve, see the Appendix.

Computing an induced curve of shortest length equals the NP-hard problem of finding the shortest length Hamiltonian path in the complete distance graph. However there exist a PTAS to approximate the optimal solution [7]. The approximation obtained is not necessarily a simple path. However, the obtained curve can be transformed into a simple curve of shorter length as shown by van Leeuwen et al [15]. They prove that $O(|T|^3)$ untangling operations suffice. As one can find an intersection in $O(|T|\log|T|)$ time [14], the total runtime of the untangling process is $O(|T|^4\log|T|)$.

Given the partition $\mathcal{M}$ computed by Algorithm 1, we obtain representative point sets T_S and T_R by choosing for all i one point $s \in S_i$ and one point $r \in R_i$. For the point set T_S and T_R, we compute simple induced curves $c(T_S)$ and $c(T_R)$ as described above. Finally, we choose the curve of minimal length as the representative c_{rep}.

Now, we can construct induced curves $c(S)$, $c(R)$ with $\delta_{dF}(c(S), c(R)) = \varepsilon$, $\delta_{dF}(c(S), c_{rep}) \leq \varepsilon$ and $\delta_{dF}(c(R), c_{rep}) \leq \varepsilon$ by connecting the points in the order given by their projection on c_{rep}. Note that $c(S)$ ($c(R)$) is simple if T_S (T_R) is the vertex set of c_{rep}. The other curve may contain crossings. In general, a shortest curve through S may induce crossings on an eligible curve though R and vice versa, see Figure 8 (Appendix).

Theorem 2 *Given two point sets $S, R \subset \mathbb{R}^2$, two induced curves $c(S)$, $c(R)$ with $\delta_{dF}(c(S), c(R)) = \delta_{dF}(S, R)$ and a simple and short representative c_{rep} can be constructed in $\mathcal{O}((\min\{m,n\})^4\log(\min\{m,n\}))$ time. The representative c_{rep} is a representative of either S or R, where each coupling is represented by only one point, and c_{rep} is a $(1+\delta)$ approximation of the shortest possible curve on these points, for a $\delta > 0$.*

3 Fréchet Distance Between Point Sets

Detecting common routes of two entities or finding similar linear formations of two groups of entities based on the discrete Fréchet distance is quite restrictive. For instance, consider the simple movement of two entities moving along a straight line. Although the movement is the same, the discrete Fréchet distance between the point sets can be larger, depending on the sampled points along the line. The continuous variant corrects this shortcoming. Except for the start- and endpoints, the sampled points do not necessarily need to have a ε-correspondence in the other point set.

For the Fréchet distance between two point sets, we first observe that in contrast to the discrete variant it might not be possible to untangle at least one curve if a solution exists. For instance, imagine both point sets have points in the four corners of a square, and one point set also has points close to the two (crossing) diagonals.

Next, we show that deciding or even approximating the Fréchet distance between two point sets is an NP-hard problem.

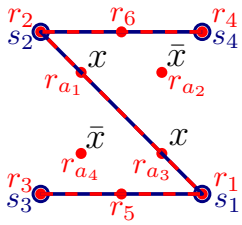


Figure 2: True assignment of the variable x.

3.1 NP-hardness

We reduce from the following SAT-variant, which was proven to be NP-hard by Berman et al. [8]. Accisano et al. [1] used the same SAT-variant to reduce from in order to show NP-hardness for the continuous unique variant of a given polygonal curve and a point set.

Problem 2 ***((3,B2)-SAT)*** *For a boolean formula with clauses of size 3, the 3-SAT problem asks if there is an assignment to the variables that satisfies the formula. The variant, denoted (3,B2)-SAT, restricts the input to formulas where each literal occurs exactly twice.*

Theorem 3 *For any $c > 1$, there is no polynomial time c-approximation algorithm for the Fréchet distance between two point sets $S, R \subset \mathbb{R}^2$, unless P=NP.*

Proof. [Proof Sketch] We construct point sets representing the variables and clauses of a given (3,B2)-SAT) instance. The clause-points have no ε-neighbor in the other point set and are within a sufficiently small distance only to segments that define an assignment of a variable. By construction, there is segment for each clause point within a sufficiently small distance, if and only if the corresponding assignments of the variables satisfy the formula. An example of the correspondence between a variable assignment and the constructed point sets is illustrated in Figure 2. A detailed proof can be found in the Appendix. □

Despite the hardness of the problem, we show that we can use the given geometric structure to solve the problem much more efficiently compared to the naive brute-force method of trying all pairs of induced curves on S and R which runs in $\mathcal{O}(n!m!)$ time.

3.2 Exponential Time Algorithm

First, we introduce the notion of *floating* and *anchored* points and *visitor* segments. In the following, (indexed) points s are always points from S and (indexed) points r are always points from R.

Definition 3 *For two point sets $S, R \subset \mathbb{R}^2$ and a value $\varepsilon > 0$, a point s is a* floating point *if there is no point r, such that $d(s,r) \leq \varepsilon$. Otherwise, s is called an* anchored *point. Furthermore, we say a floating point $s \in S$ is* isolated, *if there are no two points r_1, r_2 such that $d(s, \overline{r_1 r_2}) \leq \varepsilon$. We call a segment $\overline{r_1 r_2}$ with distance at most ε to a floating point s a* visitor *of s.*

The runtime of the algorithm we develop in this section is exponential in the number of floating points and incorporates the maximum number of visitors for a floating point. Basically, for all feasible points-visitor combinations, we efficiently check if a combination can be extended to curves realizing the Fréchet distance for the given value ε. First, we consider some simple cases.

Observation 1 *If all points in S and R are anchored points, the discrete Fréchet distance and therefore the Fréchet distance between the point sets is smaller than ε and we can compute the induced curves as shown in Section 2. Furthermore, if there are isolated points in S or in R or if there is no anchored pair, we can conclude that $\delta_F(S,R) > \varepsilon$.*

Hence in the following, we assume that the set of floating points and the set of anchored points in S and R are not empty and no point is isolated.

We start by preprocessing the input data and subsequently iterate over all possible points-visitor combinations where each floating point is assigned a visitor. Hence, a points-visitor combination consist of subgraphs of the complete graphs on S and R. For each such combination, we test whether it can be extended to induced curves with Fréchet distance at most ε, and discard it whenever we determine that it cannot. We do so in three main steps:

1. Test if a points-visitor combination is *pre-valid*.
2. Test if a pre-valid combination can be extended to a *pre-valid subcurve collection*.
3. Test if a pre-valid subcurve collection can be extended to induced curves realizing the Fréchet distance subject to the value $\varepsilon > 0$.

Preprocessing To determine the floating points and the anchored points, we compute and store for each point $s \in R$ a list $L_s = \{r \in R \mid d(s,r) \leq \varepsilon\}$. Furthermore, we compute and store a list M_s of all segments of R within ε-distance to s. Analogously, we store such lists for all points $r \in R$. We set a pointer from each anchored point s to its entry in the list L_{r_i} for all lists $L_{r_i} \ni s$. Furthermore, we set a pointer from s to each pair of anchored points $(s, \hat{s})$, $s \neq \hat{s} \in S$ and a pointer from each anchored point pair to its entry in the lists M_{r_i}, if the pair is contained in M_{r_i}. We set pointers analogously for points in R. Computing all list and thus the preprocessing step takes $\mathcal{O}(nm^2 + n^2m)$ time.

Let a_S be the number of floating points in S and let a_R be the number of floating points in R. Furthermore, for every floating point x of S or R, let k be the maximal number of visitors of x. Each of the $\mathcal{O}(k^{a_s+a_R})$ points-visitor combinations consist of all floating points in $S \cup R$ and one segment of M_x for each floating point x.

Step 1 We define *pre-valid* and an *invalid* combinations and discard invalid combinations.

Definition 4 *A points-visitor combination C is* pre-valid *if all points in C have degree at most 2 and if C does not contain a circle. Furthermore, for each floating point-visitor pair $(s, \overline{r_1r_2}) \in C$, r_1 is either an anchored point or, if r_1 is a floating point with visitor $\overline{s_1s_2}$,*

- ***Case 1:*** *either $(s_1, \overline{r_1r_2}) \in C$ or s_1 is an anchored point within ε-distance to $\overline{r_1r_2}$ or*
- ***Case 2:*** *either $(s_2, \overline{r_1r_2}) \in C$ or s_2 is an anchored point within ε-distance to $\overline{r_1r_2}$.*

The same must hold for r_2. If a combination C is not pre-valid, we say C is invalid.

Lemma 4 *An invalid points-visitor combination cannot be extended to obtain two induced curves with Fréchet distance at most ε.*

Proof. Obviously, if C contains a circle or a point of degree > 3, C cannot be extended to induced curves. Let r_1 be a floating point, $m(s)$ be the projection of s onto $\overline{r_1r_2}$ and $m(r_1)$ the projection of r_1 onto $\overline{s_1s_2}$. As $\overline{r_1r_2}$ is the visitor of s and $\overline{s_1s_2}$ is the visitor of r_1 in C, either $\delta_F(\overline{m(s)r_1}, \overline{s_1m(r)}) \leq \varepsilon$ or $\delta_F(\overline{m(s)r_1}, \overline{s_2m(r)}) \leq \varepsilon$. Thus, either **Case 1** or **Case 2** must occur. The argumentation is analogue for r_2. □

Step 2 We now proceed with a pre-valid points-visitor combination C. Let $\overline{r_1r_2}$ be a visitor in C and let $S_{\overline{r_1r_2}} = \{s_a, s_b, \ldots, s_l\}$ be the set of all floating points of S visited by $\overline{r_1r_2}$ in C. We consider the cases where r_1 is floating or anchored. When a point x is added as an interior point of a subcurve, we delete x from all lists L and the corresponding segments from all lists M.

Let r_1 be a floating point with visitor $\overline{s_1s_2}$ and let s_1 be a floating point, that is $s_1 \in S_{\overline{r_1r_2}}$. Then an induced curve on S with Fréchet distance at most ε to an induced curve on R must visit s_1 before any other point in $S_{\overline{r_1r_2}}$. We analogously determine a point last visited by the segment $\overline{r_1r_2}$, if r_2 is also a floating point such that its visitor's endpoint is a floating point. We order the remaining points of $S_{\overline{r_1r_2}}$ along $c(S)$ in between s_1 and s_3 by their projection along $\overline{r_1r_2}$, see Figures 3,4,5,6.

If s_1 is an anchored point, we order the points in $S_{\overline{r_1r_2}}$ as in the case of s_1 being a floating point. Let r_{s_1} be the closest point to s_1 in R and let $s_\alpha \in S_{\overline{r_1r_2}}$ be the first

Figure 3: Floating point and visitor, (left) with anchored endpoints and (right) with one floating endpoint.

Figure 4: (right) a visitor with two floating endpoints and (left) two floating points with the same visitor.

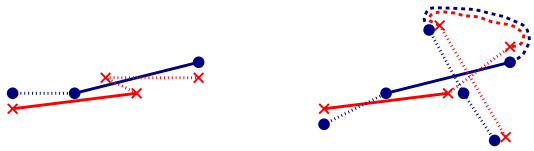


Figure 5: (left) Connecting both points visited by the same segment induces a zag; (right) Point within ε-distance to a vistors's endpoint can be currently ignored.

point visited by $\overline{r_1r_2}$ after s_1. Now we need to handle several cases: If there are anchored points s_p, s_q of S not in C such that $d(r_{s_1}, \overline{s_ps_q}) \leq \varepsilon$, we do not need to handle r_{s_1} now as we can visit r_{s_1} later via $\overline{s_ps_q}$, see Figure 5(right). Similarly, we do not need to handle r_{s_1} if there is another point $s_t \in S$, $s_t \notin C$ within ε distance to r_{s_1}, see Figure 6(left). If both of these possibilities to visit r_{s_1} later do not exist, we need to incorporate r_{s_1} on the curve $c(R)$ in between r_1 and

- the point of R ($\neq r_1$) with visitor $\overline{s_1s_2}$ in C such that its projection onto $\overline{s_1s_2}$ is closest to s_1 among all points of R visited by $\overline{s_1s_2}$ in C or
- the nearest neighbor of s_2 in R if r_1 is the only point of R visited by $\overline{s_1s_2}$ in C.

This is possible, if and only if $d(r_1, r_{s_1}) \leq \varepsilon$. If $d(r_1, r_{s_1}) > \varepsilon$, we discard C from further consideration, see Figure 6(right). If we do not need to handle r_{s_1} immediately and $d(r_1, r_{s_1}) \leq \varepsilon$, we store r_{s_1} in a list Π of points possibly to be added later.

Now if r_1 is an anchored point, we similarly order the points of $S_{\overline{r_1r_2}}$ along their projection along $\overline{r_1r_2}$ and connect the first point of that order with a point in S that belongs to the same component as r_1 of a pre-computed $\varepsilon-$coupling of the anchored points, see Figure 3(left). The same constructions are executed for r_2, and analogously, we construct subcurves on the point set R. In a last step, we consider each point $x \in \Pi$ and incorpo-

Figure 6: Point within ε-distance to a vistor's endpoint must be added and induces a zag.

rate x on the curve as described above, if $|L_x| = 0$ and $|M_x| = 0$.

For each such subcurve construction, we need to verify that the subcurves are within Fréchet distance at most ε. Let $s_\alpha \in S_{\overline{r_1r_2}}$ be the first point visited by r_1r_2 after s_1 and let $s_\omega \in S_{\overline{r_1r_2}}$ be the point visited by r_1r_2 before s_3. If s_1 and s_α or s_ω and s_3 induce a zag such that the Fréchet distance of this zag and the visitor $\overline{r_1r_2}$ exceeds ε, we discard C from further consideration. See Figure 5(left) for an example. The same checks are performed for the constructed subcurves on R.

If C is not discarded, we finish step 2 with a collection of subcurve pairs on S and R, each with Fréchet distance at most ε. In particular, all subcurves start and end at anchored points. We denote such a collection as a *pre-valid* subcurve collection.

Step 3 Given a pre-valid subcurve collection SC, we define ΔS and ΔR as the anchored points of S and R not in SC. A point $s \in \Delta S$ is *matchable*, if $|L_s| > 0$ after performing step 2. That is, if there is a point $r \in \Delta R$ or a point r in SC of degree one within ε-distance to s. We say r is a *free ε-partner* of s. If $|L_s| = 0$, but $|M_s| > 0$, that is if there is a segment g between two free ε-partners in R or between two degree-one points of SC or between a degree-one point of SC and a free ε-partner such that $d(s, g) \leq \varepsilon$, we say s is *visitable*. Note that all other segments in $|M_s|$ have been deleted during the previous steps. These notations are also used for points in ΔR.

By definition, there exists a discrete Fréchet coupling for the union of the matchable points of ΔS, ΔR and the points of SC with degree one. By taking care of the order of the components of the coupling, we can ensure that all visitable points are visited: Let $g = \overline{r_1r_2}$ be the segment with ε distance to the visitable point s. If a coupling exists, there is a segment $\overline{s_1s_2}$ such that $d(r_1, s_1) \leq \varepsilon$ and $d(r_2, s_2) \leq \varepsilon$. Thus, we can bend $\overline{s_1s_2}$ around s and have $\delta_F(\overline{r_1r_2}, (s_1, s, s_2)) \leq \varepsilon$. An eligible order can be extracted from the lists M of the visitable points. Therefore, the following lemma holds:

Lemma 5 *Given a pre-valid subcurve collection SC on the point sets S and R. The subcurves of SC can be concatenated to induced curves $c(S)$, $c(R)$ such that $\delta_F(c(S), c(R)) \leq \varepsilon$, if and only if each point $s \in \Delta S$ ($r \in \Delta R$) is either matchable or visitable.*

Hence, if the conditions of Lemma 5 hold, we can construct two curves with Fréchet distance at most ε as described for the discrete variant. The following theorem sums up the results of this section.

Theorem 6 *Given two point sets $S, R \subset \mathbb{R}^2$ of size $|S| = n$, $|R| = m$, a value $\varepsilon > 0$ and lists L_x, M_x for all points in $S \cup R$ as defined above, deciding if $\delta_F(S, R) \leq \varepsilon$ takes $\mathcal{O}\left(k^a((m + n - a) + a \log a)\right)$ time, where a is the number of floating points in S and R and k is the maximum number of visitors for a floating point.*

Proof. Correctness follows from Lemma 5 and the description of constructing the subcurves based on a given points-visitor combination. Note that during the subcurve construction, the inserted segments are determined by the specific points-visitor combination. The only freedom of choice during the construction is how to connect one point to another point within ε-distance to an anchored point. The rule of choosing a point of within the same component of an ε-coupling prevents from deleting a point that possibly is the only point within ε-distance to another anchored point p not considered yet. That is, if $\delta_F(R, S) \leq \varepsilon$, we eventually process a points-visitor combination that can be extended to two curves with Fréchet distance at most ε.

For each combination C, we need to process all floating points and possibly $2a$ anchored points to construct SC (one floating point can be connected to at most two anchored points). The processing consists of projecting, sorting and measuring the Fréchet distance of curves of length 3 (checking the induced zags). This takes $\mathcal{O}(a \log a)$ time. Processing the list Π takes linear time in the number of floating points. Each time an anchored point is used as an interior point of a subcurve, we delete the corresponding entries of L and M using the pointers in constant time. During the construction of SC we only process $\mathcal{O}(a)$ anchored points, thus updating the lists takes $\mathcal{O}(a)$ time. Once a pre-valid subcurve collection is found, the condition of Lemma 5 can be verified in $\mathcal{O}(m+n-a)$ time checking the size of the lists of points in ΔS and ΔR. □

4 Conclusion

For the discrete Fréchet distance between two point sets, we showed that we can construct a simple representative curve by approximating the shortest Hamiltonian path problem on the points given by a partition of the two point sets. For the continous variant, we reduced the possibilities of constructing the induced curves by carefully exploiting all given geometry. The algorithmic approach presented here takes into account the number of floating points and number of segments that are close to a floating point. That is, if there is a low number of such segments, the algorithmic approach can be applied even if the number of floating points is large.

References

[1] P. Accisano and A. Üngör. Approximate matching of curves to point sets. In *Proc. Canadian Conference on Computational Geometry*, 2014.

[2] P. Accisano and A. Üngör. Finding a curve in a point set. *CoRR*, abs/1405.0762, 2014.

[3] P. Accisano and A. Üngör. Matching curves to imprecise point sets using Fréchet distance. *CoRR*, abs/1404.4859, 2014.

[4] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. In *Proc. 24th Annual Symposium on Discrete Algorithms*, pages 156–167, 2013.

[5] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes (extended abstract). In *Proc. 7th Annual Symposium on Computational Geometry*, page 186–193, 1991.

[6] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1&2):75–91, 1995.

[7] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 37th Annual Symposium on Foundations of Computer Science*, page 2, 1996.

[8] P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity*, 2003.

[9] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *55th Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014.

[10] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017.

[11] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical report, 1994.

[12] T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997.

[13] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Staying close to a curve. In *Proc. Canadian Conference on Computational Geometry*, 2011.

[14] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. 17th Annual Symposium on Foundations of Computer Science*, pages 208–215, 1976.

[15] J. van Leeuwen and A. A. Schoone. Untangling a travelling salesman tour in the plane. In *Proc. 7th Conference Graph-Theoretic Concepts in Computer Science*, pages 87–98, 1981.

[16] T. Wylie and B. Zhu. Following a curve with the discrete Fréchet distance. *Theoretical Computer Science*, 556:34–44, 2014.

Appendix

Untangling a curve

A curve $c(T)$ of shortest length induced by a point set T also fulfills the crossing free property: Let e and f be two crossing segments as shown in Figure 7 and let $l(e)$ be the Euclidean length of the segment e. Then, by the triangle inequality, we have $l(e_1)+l(f_1) > a$, $l(e_2)+l(f_2) > b$ and $l(e_1)+l(f_2) > a$, $l(e_2)+l(f_1) > b$ and therefore $l(e)+l(f) > l(a)+l(b)$. Hence untangling a curve leads to a shorter curve.

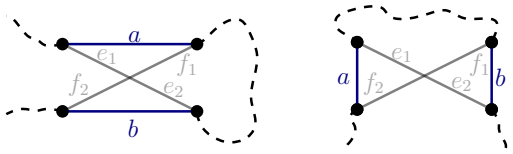


Figure 7: Replacing two crossing segments, e and f by two segments, a and b, of shorter total length.

No short crossing free curves

For two point sets S, R and a matching realizing their discrete Fréchet distance, a shortest curve through one of the point sets may induce crossings in the other and vice versa.

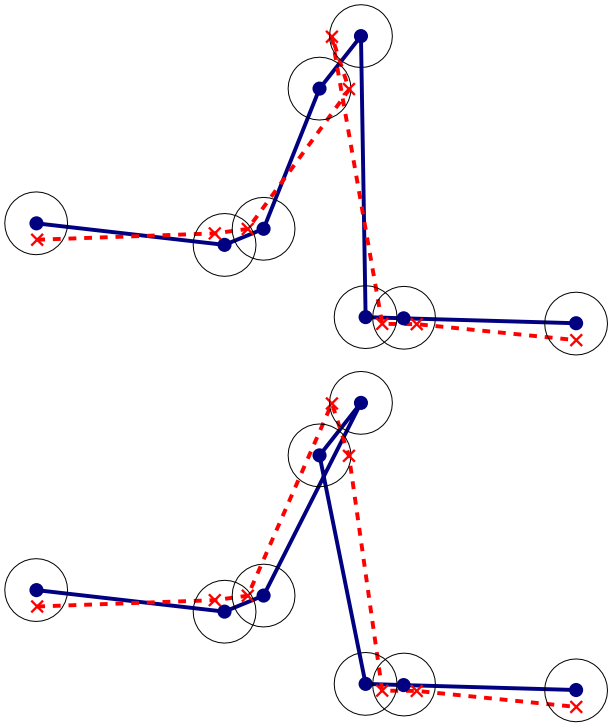


Figure 8: Two point sets S, R (indicated by crosses, dots), which do not have crossing free shortest realizations $c(S), c(R)$.

Proof of Theorem 3

We assume that no two clauses have two literals in common. Note, that for any formula that violates this assumption, an equivalent (3,B2)-SAT instance fulfilling the assumption can be constructed in polynomial time [8].

Similar to [1], the core of the construction are points representing clauses of a (3,B2) instance that all must be visited in order to satisfy the underlying SAT formula. In our proof, the clause-points are part of the vertex set of $c(R)$. We prove that for any input variable x, there is either one segment of $c(S)$ that passes all positive literal's clauses within ε-distance and no segment exists passing all negative literal's clauses within ε-distance, or vice versa. Otherwise, $c(S)$ cannot be within Fréchet distance at most ε to any induced curve on R.

Construction of the Point Sets Given a (3,B2)-SAT instance A, we construct two point sets S and R such that A is satisfiable if and only if a $\delta_F(S,R) \leq \varepsilon$, for $\varepsilon > 0$ to be computed below. For each clause a of A, we embed one point r_a of R in $\mathbb{R}^2$. Let P be the set of points r_a for all clauses a. We refer to these points as *clause-points*. As each literal l occurs exactly twice in A, l defines a line through two points of P, namely through the two points corresponding to the clauses where the literal occurs. Let x be a variable of A and let r_{a_i}, $i = 1,2,3,4$ be the points of $P \subset R$ that correspond to the clauses where x and $\bar{x}$ occur. These points must be embedded such that no three points are colinear. For instance, we can embed them along a (fraction of a) circle. We place two points s_1, s_2 of S and two points r_1, r_2 of R along the line defined by x. The points s_1 and r_1 and the points s_2 and r_2 have the same coordinates, respectively. Furthermore, the corresponding points r_{a_i} lie between s_1 and s_2. Analogously, we place points s_3, s_4, r_3 and r_4 on the line defined by $\bar{x}$. We refer these points as *variable-points*. To complete the construction of the variable gadget, we place one point r_5 of R on the segment between s_1 and s_3 and one point r_6 of R on the segment between s_2 and s_4 and refer these points as *segment-forced-points*. See Figure 2 for an example of a variable gadget.

The core idea of the construction is that any curve induced by S being a witness of $\delta_F(S,R) \leq \varepsilon$ must contain either the segment s_1s_2 or the segment s_3s_4, each of which defines an assignment of the corresponding variable.

During the construction of the sets S and R, we need to ensure that no clause-point and two variable-points that correspond to two different literals are colinear. Analogously, the construction must exclude any colinearity between a segment-forced-point and two variable-points of different variables. To this end, we embed all n clause-points on n consecutive corners of a regular $4n$-gon. Subsequently, we iteratively embed the variable- and segment-forced-points maintaining the *non-colinearity* property described above by computing and avoiding the coordinates of points which violate this property.

All lines defined by a literal pass the first, second and third quadrant, where the quadrants are defined by the perpendicular lines through the leftmost and rightmost clause-point. We start by inserting variable-points of R and S in the first quadrant. We randomly chose one variable, say x, insert two variable-points s_1, s_2 of S and r_1, r_2 of R along the corresponding literal-lines and insert one point r of R on the line segment s_1s_2. Next, we compute all intersections of the lines defined by s_1 and r and by s_2 and r, respectively, with all lines defined by any literal other than x or $\bar{x}$. We maintain these intersection points as *forbidden coordinates* for inserting further variable-points in a list F. We progress with inserting variable- and segment-forced points in the first quadrant and update the list of forbidden coordinates after each insertion.

After the insertion of all points of R and S in the first quadrant, we compute all intersections of lines defined by a variable-point and a clause-point where the corresponding literal of the variable-point does not occur, with all lines defined by the literals. We add these points to F and start inserting variable- and segment-forced points in the third quadrant such that these coordinates are not contained in F, again, continuing to update F after each insertion.

Once we have constructed the point sets S and R, we compute a value $\hat{\varepsilon}$, as the minimum of the minimal distance of any clause-point of R to a segment defined by two points which correspond to two different literals and the minimal distance of any segment-forced point to a segment defined by two points corresponding to different variables. Note that $\hat{\varepsilon} > 0$, as we excluded colinearity in all these cases. Now, we define $\varepsilon = \frac{\hat{\varepsilon}}{c}$.

Let n be the number of variables in A. It is easy to see that the construction time is polynomial in n: The number of computed intersections and the total number of inserted points are polynomial in n and computing ε runs in polynomial time as there is a quadratic number of pairs of variable points and a linear number of segment-forced- and clause-points.

Reducing from (3,B2)-SAT We show that a (3,B2)-SAT formula A is satisfiable if and only if $\delta_F(S,R) \leq \varepsilon$. We say a point of r is *visited* by a segment s of $c(S)$ if the distance of r to s is at most ε.

First, let $c(S)$ and $c(R)$ be two induced curves such that $\delta_F(c(S),c(R)) \leq \varepsilon$. Clearly, every point of R must be visited by a segment of $c(S)$, and, vice versa, every point of S must be visited by a segment of $c(R)$. Each subcurve of S on the points of a variable gadget, say for the variable x, must contain the two unique segments which cover the segment-forced-points of R corresponding to the current variable. Now consider the two pairs of variable-points of S, each consists of the two points corresponding to the same literal. Let us denote these pairs as (s_{x_1}, s_{x_2}) and $(s_{\bar{x}_1}, s_{\bar{x}_2})$. Obviously, both segments $s_{x_1}s_{x_2}$ and $s_{\bar{x}_1}s_{\bar{x}_2}$ cannot be part of the curve as in this case these segments together with the segments forced by the segment-forced points form a circle and thus $c(S)$ is not an induced curve. As by construction, no clause-point can be visited by a segment between points corresponding to different literals, any solution must chose one of the two possible segments defined by the variable gadget. Note that no two clauses have two literals in common and therefore the claim follows from an easy counting argument. The segment contained in the solution induces an assignment of the variable x and as all clause points are visited, A is satisfied by the assignments induced by $c(S)$. There is some freedom of how the congruent segments of $c(R)$ of the segments defining an assignment are subdivided by the clause-points which have distance at most ε (and

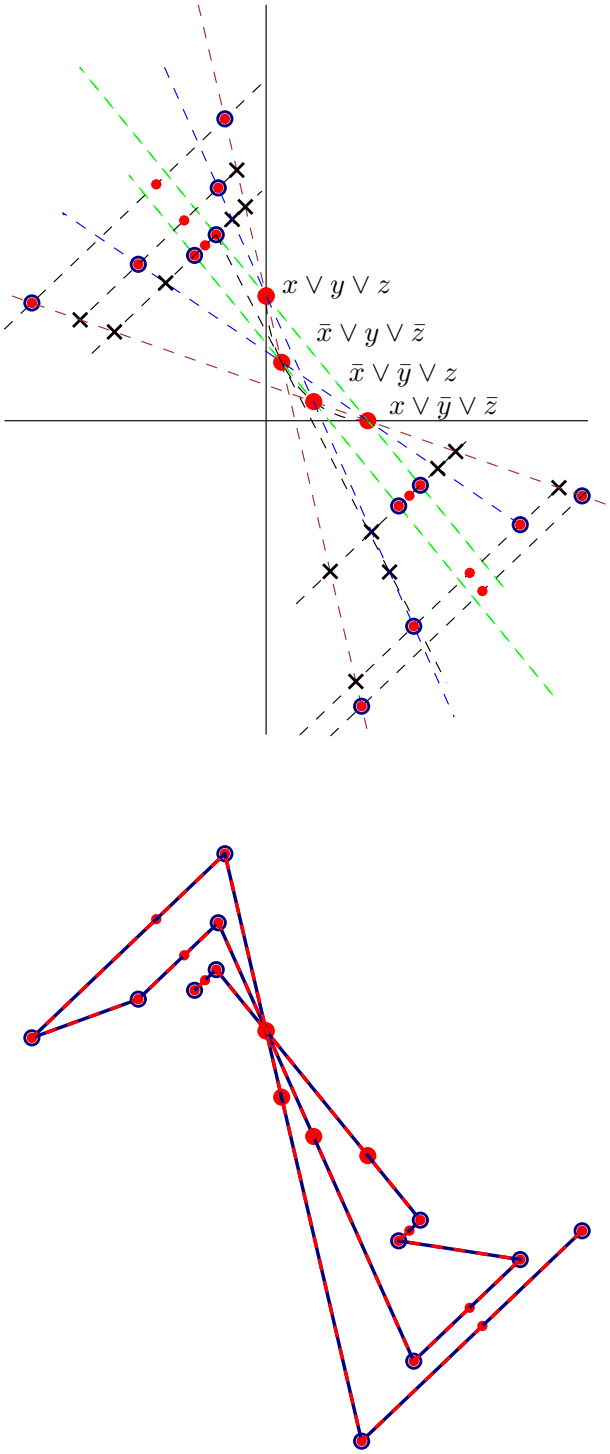


Figure 9: Point set S marked in blue and point set R marked in red for the (B2,3)-SAT Formula $A = (x \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (x \vee \bar{y} \vee \bar{z})$. Some of the points in L are shown as crosses. For the blue curve $c(s)$ and the red curve $c(R)$, we have $\delta_F(c(S), c(R)) = 0$ and the induced assignment for the variables - positive for all variables - satisfies A.

by construction distance 0) to a segment. As we consider the unique case, all clause-points can be added only once and some clause points have distance 0 to several of these segments. However, the assignment of the variable is independent on how the clause-points are contained.

Second, let α be a satisfying assignment of the formula A. For each variable gadget, chose the segment between two variable-points of S which correspond to the assignment of the variable given by α. Furthermore, add the segments defined by the two variable-points in the first quadrant and the two variable-points in the third quadrant to $c(S)$. Finally, connect the subcurves on points of the variable gadget arbitrarily to obtain the complete curve $c(S)$ on all point of S. Each segment of $c(S)$ induces a segment of $c(R)$. If a segment of $c(S)$ connects two variable-point which refer to different variables, then the segment connecting the congruent points of R is added to $c(R)$. Otherwise, if a segments g of $c(S)$ connects two points which refer to the same variable but different literal, we add a subcurve of $c(R)$ with congruent start and endpoints as g, both adjacent to the corresponding segment-forced-point. If g connects variable points which refer to the same literal, we add the subcurve which consist of the congruent points of g in R and the corresponding clause-points. Here, we add each clause point only once on $c(R)$. As $\delta_F(c(S), c(R)) = 0 < \varepsilon$ the claim follows.

Figure 9 shows an example construction for a simple (B2,3)-SAT instance.

Realizing an m-uniform four-chromatic hypergraph with disks

Gábor Damásdi
MTA-ELTE Lendület
Combinatorial Geometry Research Group

Dömötör Pálvölgyi
MTA-ELTE Lendület
Combinatorial Geometry Research Group

Abstract

We prove that for every m there is a finite point set $\mathcal{P}$ in the plane such that no matter how $\mathcal{P}$ is three-colored, there is always a disk containing exactly m points, all of the same color. This improves a result of Pach, Tardos and Tóth who proved the same for two colors. The main ingredient of the construction is a subconstruction whose points are in convex position. Namely, we show that for every m there is a finite point set $\mathcal{P}$ in the plane in convex position such that no matter how $\mathcal{P}$ is two-colored, there is always a disk containing exactly m points, all of the same color. We also prove that for unit disks no similar construction can work.

1 Introduction

Coloring problems for hypergraphs defined by geometric range spaces have been studied a lot in different settings. A pair $(\mathcal{P}, \mathcal{S})$, where $\mathcal{P}$ is a set of points in the plane and $\mathcal{S}$ is a family of subsets of the plane (the *range space*), defines a (primal) hypergraph $\mathcal{H}(\mathcal{P}, \mathcal{S})$ whose vertex set is $\mathcal{P}$ and for each $S \in \mathcal{S}$ we add the edge $S \cap \mathcal{P}$ to the hypergraph. Given any hypergraph $\mathcal{G}$, a planar realization of $\mathcal{G}$ is defined as a pair $(\mathcal{P}, \mathcal{S})$ for which $\mathcal{H}(\mathcal{P}, \mathcal{S})$ is isomorphic to $\mathcal{G}$. If $\mathcal{G}$ can be realized with some pair $(\mathcal{P}, \mathcal{S})$ where $\mathcal{S}$ is from some family $\mathcal{F}$, then we say that $\mathcal{G}$ is realizable with $\mathcal{F}$.

A hypergraph is (properly) c-colorable if its vertices can be colored by c colors such that no edge is monochromatic. In this paper we focus on the c-colorability of hypergraphs realizable with disks. It is an easy consequence of the properties of Delaunay-triangulations and the Four Color Theorem that any hypergraph realizable with disks is four-colorable if every edge contains at least two vertices. Since K_4 is realizable with disks, this is sharp. But are less colors sufficient if all edges are required to contain at least m vertices for some large enough constant m? Pach, Tardos and Tóth [19] have shown that two colors are not enough for any m, i.e., for any m, there exists an m-uniform hypergraph that is not two-colorable and that permits a planar realization with disks. Our main theorem is the following strengthening, which shows that three colors are also not enough by realizing a four-chromatic hypergraph, completely resolving this question.

Theorem 1 *For any m, there exists an m-uniform hypergraph that is not three-colorable and that permits a planar realization with disks.*

The proof of Theorem 1 is based on two ideas. The first one is from [10], where colorings of so-called ABAB-free hypergraphs were considered. We will use a construction, based on one from [10], to create a point set that is not two-colorable with respect to disks and, furthermore, the points are close to a prescribed set of points that lie on a circle. Then using ideas from [1] and [19], we will combine several copies of this non-two-colorable construction to create point sets that are not three-colorable. The non-two-colorable construction can be of independent interest.

Theorem 2 *For any m, there exists an m-uniform hypergraph that is not two-colorable and that permits a planar realization with disks that all contain some fixed point.*
Moreover, in the realization $(\mathcal{P}, \mathcal{D})$, the points $\mathcal{P}$ can be placed arbitrarily close to some given points on a circle such that the boundary of each disk from $\mathcal{D}$ is also arbitrarily close to this circle.

Previously, such a construction was only known for pseudo-disks containing a fixed point, and it was also shown that such hypergraphs are always three-colorable (already for $m = 2$) [1].

Note that if we require the points of $\mathcal{P}$ to be placed close enough to the given points on the circle, then the points of $\mathcal{P}$ will be in convex position.

To complement our results, we also show that no similar construction for unit disks exists.

Theorem 3 *For any k, any finite point set $\mathcal{P}$ can be k-colored such that any unit disk, that contains some fixed point $o \notin \mathcal{P}$ and $8k-7$ points from $\mathcal{P}$, will contain all k colors.*

It is known that without requiring the common point o, the statement does not hold [17].

By using the well-known equivalence of the hypergraphs defined by primal and dual range spaces of the translates of any set [16, 18], we can conclude the following.

Corollary 4 *For any k, any finite collection of unit disks $\mathcal{D}$ can be partitioned into k parts such that any point of a fixed unit disk $D_0 \notin \mathcal{D}$ that was covered by at least $8k-7$ members of $\mathcal{D}$ will be covered by all k parts.*

This statement is sharp in the sense that for a disk D_0 of larger radius it fails already for $k = 2$, even if the members of $\mathcal{D}$ are required to be very close to each other; this follows from taking the dual of the construction from [19]. The function $8k - 7$ is unlikely to be sharp.

The rest of the paper is organized as follows. For $1 \leq i \leq 3$, Theorem i is proved in Section $(3i^2 - 11i + 14)/2$, while in Section 5 we make some concluding remarks, and we end this Introduction with a bit more history, and a basic observation.

Pach proved in 1986 that any sufficiently thick covering of the plane [16] by the translates of a centrally symmetric open convex polygon can be partitioned into two disjoint coverings. The proof followed from showing that for every such polygon P there is an $m(P)$ for which any hypergraph, whose edges contain at least $m(P)$ vertices and can be realized with translates of P, is two-colorable. Several results followed, eventually showing that the similar statement is true for the translates of all convex polygons [7, 21], while counterexamples were given for non-convex polygons [19, 20] and convex shapes with a smooth boundary [17]. We know much less when instead of translates, homothetic copies are considered [2, 5, 9, 13] or about the problem of decomposing into multiple coverings/polychromatic colorings [4, 7, 22]. For more results, see the decade-old survey [18] or the webpage `https://coge.elte.hu/cogezoo.html`.

The above papers mainly focused on two- or polychromatic colorability. Proper three-colorability of geometric hypergraphs was studied in detail in [8] and [11]. In the latter paper it was shown that for every convex polygon P there is an $m(P)$ such that every $m(P)$-uniform hypergraph realizable by homothetic copies of P is proper three-colorable. Our results imply that this is not the case for disks, disproving a conjecture from both of the above papers.

In this paper all disks are assumed to be open.[1] Let $\mathcal{P}$ be a point set and let $\mathcal{D}$ be a family of disks. An important folklore observation that we will use many times is that small perturbations of the points and the disks will not change the hypergraph $\mathcal{H}(\mathcal{P}, \mathcal{D})$. To put this into more precise terms, we will say that two points are *ε-close* if their distance is less than ε and two disks/circles are *ε-close* if their centers are ε-close and the difference of their radius is also smaller than ε.

[1] But note that our results also hold for closed disks, as we could slightly shrink any finite system of open disks to contain the same points of a finite point set.

Observation 1 *Suppose we have a finite point set $\mathcal{P}$ and a finite set of open disks $\mathcal{D}$ such that none of the points lie on the boundary of any of the disks. Then there is an ε such that replacing each disk with any ε-close disk and each point with any ε-close point will not change the hypergraph $\mathcal{H}(\mathcal{P}, \mathcal{D})$.*

2 A point set that is not two-colorable

Here we prove Theorem 2 by realizing for any m a non-two-colorable m-uniform hypergraph with disks that all contain some fixed point.

Our main lemma is the following. Combined with Observation 1, this gives us a way to perturb the points of $\mathcal{P}$ with changing only a small part of the hypergraph $\mathcal{H}(\mathcal{P}, \mathcal{D})$.

Lemma 5 *If $\varepsilon > 0$, C is circle, and $a, b_1, \ldots, b_t, c$ are points on C in this order, then there is a circle C' and points $b'_1, \ldots, b'_t$ on C' with the following properties.*

1. *C' is ε-close to C.*
2. *b'_i is ε-close to b_i for each $i \in [t]$.*
3. *C' intersects C between a and b_1, and between b_t and c.*
4. *Each b'_i is outside of C.*

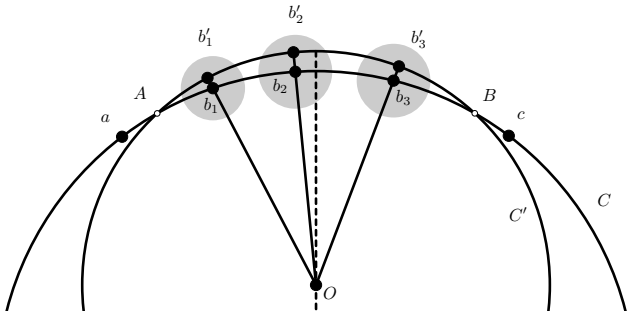


Figure 1: Lemma 5.

Proof. Choose points A and B between between a and b_1 and between b_k and c, respectively (see Figure 1). Choose O on the perpendicular bisector of AB, close to the center of C. C' will be the circle centered at O passing through A and B. Project $b_1, \ldots, b_t$ onto C' using O as center. If O is close enough to the center of C, this will clearly satisfy the requirements. □

2.1 Hypergraphs based on rooted trees

The following hypergraph construction was used in [19] to create several counterexamples for coloring problems.

Definition 6 *For any rooted tree T, let $\mathcal{H}(T)$ denote the hypergraph on vertex set $V(T)$, whose hyperedges are all sets of the following two types.*

1. *Sibling hyperedges: for each vertex $v \in V(T)$ that is not a leaf, take the set $S(v)$ of all children of v.*
2. *Descendent hyperedges: for each leaf $v \in V(T)$, take all vertices along the unique path $Q(v)$ from the root to v.*

It is easy to see that $\mathcal{H}(T)$ is not two-colorable for any T. Either there is a monochromatic sibling edge, or we can follow the color of the root down to a leaf, finding a monochromatic descendent edge. We can create an m-uniform hypergraph by choosing T to be the complete m-ary tree of depth m. The non-two-colorable construction of Pach, Tardos and Tóth is also based on these hypergraphs.

Theorem 7 (Pach, Tardos and Tóth [19]) *For any rooted tree T, the hypergraph $\mathcal{H}(T)$ permits a planar realization $(\mathcal{P}, \mathcal{D})$ with disks in general position such that every disk $D \in \mathcal{D}$ has a point on its boundary that does not belong to the closure of any other disk $D' \in \mathcal{D}$.*

In order to be able to later build a point set that is not three-colorable, we will first extend Theorem 7 by showing that we can require the points to be close to a prescribed set of concyclic points and require the disks to be close to the circle that contains the prescribed points. (We loose the property that every disk $D \in \mathcal{D}$ has a point on its boundary that does not belong to the closure of any other disk $D' \in \mathcal{D}$, so it is not a generalization in the strong sense.)

Theorem 8 *If $\gamma > 0$, C is a circle and $q_1, q_2, \ldots, q_n$ are distinct points on C, then for any rooted tree T on n vertices, the hypergraph $\mathcal{H}(T)$ permits a planar realization $(\mathcal{P}, \mathcal{D})$ with disks such that*

(I) $\mathcal{P} = \{p_1, \ldots, p_n\}$, and each p_i and q_i are γ-close.

(II) Every $D \in \mathcal{D}$ is γ-close to C.

An important property of rooted trees is that we can order their vertices in a special way. For a vertex v let $Des(v)$ denote all descendants of v. Keszegh and Pálvölgyi [10] have shown that there is an ordering on the vertices of T such that

1. For each vertex $v \in V(T)$ the vertices in $S(v)$ are consecutive and they appear in the order later than v.
2. Furthermore, suppose $S(v) = \{r_1, \ldots, r_k\}$ and they are in this order. Then the vertices $r_1, \ldots, r_{k-1}, r_k, Des(r_k), Des(r_{k-1}), \ldots, Des(r_1)$ are ordered like this, and the rest of the vertices of T are not in this interval. (The internal order of each $Des(r_i)$ is not specified by this statement.)

Call an order satisfying these properties a *siblings first order* [1] of T.

Proof. [of Theorem 8] We start by showing that the sibling hyperedges can be easily realized and we only need to consider the descendent hyperedges.

Sibling hyperedges

From the properties of the sibling first order we know that for each v the vertices of $S(v)$ are consecutive, i.e., the points in $S(v)$ are neighbours along the circle C. We apply Lemma 5 to find a disk that is γ-close to C, and contains exactly the points of $S(v)$. We also ensure that no points lie on the boundary of the disk. We repeat this for each $v \in V(\mathcal{H})$, until each sibling hyperedge is realized. Let $\mathcal{D}_{Sib}$ denote the set of these disks. We apply Observation 1 to $(\mathcal{P}, \mathcal{D}_{Sib})$ to get ε_{Sib}. that is, if we move the points such that they remain ε_{Sib}-close to their original position, the disks in $\mathcal{D}_{Sib}$ will still represent the sibling hyperedges. Therefore, it is enough to show that we can realize the descendent hyperedges for any γ.

Descendent hyperedges

It is useful to realize a slightly extended hypergraph. In $\mathcal{H}(T)$, we have a descendent hyperedge for each leaf. Now we will create a descendent hyperedge for non-leaf vertices too. So let $Q(v)$ denote the path from the root to v, and for each vertex v, we will realize the hyperedge $Q(v)$. The disk realizing $Q(v)$ will be denoted by $B(v)$. Let this extended hypergraph be denoted by $\mathcal{H}'(T)$. We will realize not only $\mathcal{H}(T)$, but $\mathcal{H}'(T)$. See Figure 2 for an example where we omitted the sibling edges.

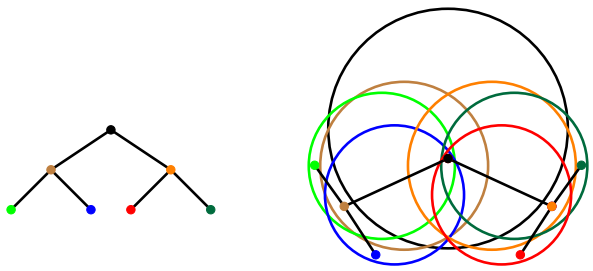


Figure 2: Disks realizing all paths from the root.

We prove the existence of the required point set by describing an algorithm that produces a solution. Let

$\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ denote the vertices of T in a siblings first order. We will create the planar realization gradually, step by step. The algorithm starts by setting $p_i = q_i$ and in each step we will update the position of some of the p_i-s. That is, we identify the vertices of the hypergraph with planar points, and we will update the position of the vertices until they realize the hypergraph $\mathcal{H}'(T)$.

During the algorithm, we need to change the position of the points many times without altering the hyperedges that we have already realized. For this reason, we introduce a set of fixed points, $\mathcal{P}_{fix}$, and a set of disks, $\mathcal{D}_{Des}$, that corresponds to the descendent hyperedges. Once a point is in $\mathcal{P}_{fix}$ we will not change its position any more. Every disk that we create will be immediately added to $\mathcal{D}_{Des}$, and we will never change its position. The unfixed points, i.e., the points in $\mathcal{P} \setminus \mathcal{P}_{fix}$, will always be kept on the boundary of $\bigcup_{D \in \mathcal{D}_{Des}} D$. Furthermore, if we add $B(v)$ to $\mathcal{D}_{Des}$ in the k-th step of the algorithm, $B(v) \cap \mathcal{P}$ will remain the same after we finished the k-th step, i.e., it will contain exactly the points of the path $Q(v)$ in its interior. Moreover, all descendants of v will be on the boundary of $B(v)$ after the k-th step, but later they will be moved off.

The structure of the algorithm is the following. We go through the vertices in the sibling first order and for each we do the following. By when we arrive at vertex p_k, the disk $B(p_k)$ representing the path $Q(p_k)$ from the root to p_k, will be already realized. For each child of p_k we will add a new disk, close to $B(p_k)$, that also realizes the same path from the root to p_k. Then we will move the points such that each new disk contains exactly one child of p_k. We have summarised the structure of the algorithm in the following pseudo code, while the phases of a step are depicted in Figure 3.

To be able to do these steps we also need a parameter δ that will ensure that the points do not move too much and the disks are close to each other. There will be only three kinds of operations that we do through the algorithm.

a) We update the position of a vertex by moving it at a distance less than the current value of δ. If it reached its final position, we add it to $\mathcal{P}_{fix}$.

b) We add a new disk to $\mathcal{D}_{Des}$ that is δ-close to one of the disks already in $\mathcal{D}_{Des}$.

c) We decrease the value of δ.

We start with $\delta = \gamma/n^2$. (The reason for this is explained later.) Every time a disk is added or the position of a vertex is changed, we use Observation 1 for $(\mathcal{P}_{fix}, \mathcal{D}_{Des})$ to update δ to a smaller value, if needed.

Algorithm 1: Structure of the algorithm

```
Set p_i = q_i, P_fix = ∅, D_Des = ∅.
Add the disk of C, B(p_1), to D_Des.
Move p_1 inside B(p_1) and add p_1 to P_fix.
for k = 1 to n do
    for each child r_i of p_k do
        Add a disk B(r_i) representing Q(p_k) to
         D_Des.            /* Using Lemma 5  */
        Move the required descendants of p_k to
         the boundary of B(r_i).
    end
    for each child r_i of p_k do
        Move r_i inside B(r_i).        /* now B(r_i)
         represents Q(r_i)  */
        Add r_i to P_fix.
    end
end
```

After any given update of δ, we only move points at distance less than δ. This ensures two things. Firstly, the points do not move far from their original position, and secondly, if we take a new disk that is δ-close to one of the disks, then it contains the same points of $\mathcal{P}_{fix}$.

The algorithm makes an initial adjustment on p_1, and then there is one step (k-th step) for each point p_k. After each step the following properties will hold.

(i) Each point p_i has either reached its final position or it lies on the boundary of the disk $B(w)$ where w is the lowest ancestor of p_i for which $B(w)$ is already defined. Also, in the second case p_i does not belong to the closure of any other disk $D' \in \mathcal{D}_{Des}$.

(ii) Suppose p_j is the parent of p_i in T. Then in the j-th step the point p_i is added to $\mathcal{P}_{fix}$ and the disk $B(p_i)$ is added to $\mathcal{D}_{Des}$.

(iii) Each disk in $\mathcal{D}_{Des}$ contains those points of P that correspond to the appropriate hyperedge of $\mathcal{H}'(T)$.

During the initial adjustment we update p_1 to lie inside C but δ-close to q_1. We add the disk corresponding to the circle C to $\mathcal{D}_{Des}$. We add p_1 to $\mathcal{P}_{fix}$ and then we update δ by applying Observation 1 for $(\mathcal{P}_{fix}, \mathcal{D}_{Des})$. Clearly properties $(i), (ii)$ and (iii) are satisfied.

In the k-th step we update the points in $Des(p_k)$. (If p_k is a leaf, we continue with the next step.) The process is depicted in Figure 3, while a detailed description can be found in the Appendix.

Finally, we need to show that we get a solution for Theorem 1. Let $\mathcal{D}$ contain the disks in $\mathcal{D}_{Sib}$ and those disk in $\mathcal{D}_{Des}$ that correspond to descendent edges that end at a leaf. Property (iii) and the argument for the sibling edges tells us that $(\mathcal{P}, \mathcal{D})$ is a planar representation of $\mathcal{H}(T)$.

For property (I) note that each point moves less than n^2 times since in the k-th step they move less than n times. We started with $\delta = \gamma/n^2$, so each point is at most $n^2\frac{\gamma}{n^2} = \gamma$ far from their original position. The first disk was given by C and each disk was taken δ-close to a previous disk, in at most n steps reaching back to the first disk. Hence, all disks are γ/n close to C, giving us property (II). □

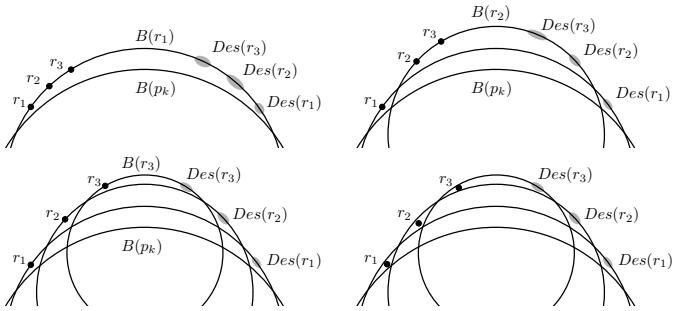


Figure 3: The k-th step.

3 A point set that is not three-colorable

Here we prove Theorem 1 by realizing for any m a non-three-colorable m-uniform hypergraph with disks.

We introduce a general operation for constructing hypergraphs that are not c-colorable, which is similar to prolonging $\mathcal{H}(T)$ with the children of a leaf. Essentially the same construction was used in [1].

Definition 9 *Suppose we have a hypergraph $\mathcal{A}$, a hypergraph $\mathcal{B}$ and let F be an edge of $\mathcal{A}$. Then we define "$\mathcal{A}$ extended by $\mathcal{B}$ through F" as follows. Take $|V(\mathcal{B})|$ copies of the edge F and add a new vertex to each copy. Then we have $|V(\mathcal{B})|$ new vertices. Add a set of edges such that they form the hypergraph $\mathcal{B}$ on these new vertices. The resulting hypergraph is denoted by $\mathcal{A} +_F \mathcal{B}$.*

Suppose $\mathcal{A}$ is m-uniform and $\mathcal{B}$ is $(m+1)$-uniform. Then, if we extend $\mathcal{A}$ by $\mathcal{B}$ through each edge of $\mathcal{A}$, we get a hypergraph that is $(m+1)$-uniform.

Claim 10 *Suppose $\mathcal{A}$ is not c-colorable and $\mathcal{B}$ is not $(c-1)$-colorable. Then any extension of $\mathcal{A}$ by $\mathcal{B}$ is not c-colorable.*

Proof. Since any coloring of $\mathcal{A}$ using c-colors has a monochromatic edge, and we have only lost the edge F, our only chance is to color the extended hypergraph such that the vertices of F are monochromatic. This implies that none of the new vertices have the same color as the vertices of F. But then the copy of $\mathcal{B}$ is $(c-1)$-colored, and thus one of its edges is monochromatic. □

Let $\mathcal{G}_i$ denote the hypergraph that has i vertices and only one edge that contains all the vertices. Clearly, $\mathcal{G}_1$ is not c-colorable for any c and $\mathcal{G}_i$ is not 1-colorable. Hence, we can build non-c-colorable hypergraphs starting from these trivial ones and using them in the extensions.

Observation 2 *For any rooted tree T the hypergraph $\mathcal{H}(T)$ can be built with a series of extensions, where each extending hypergraph is one of the $\mathcal{G}_i$-s, and we start from $\mathcal{G}_1$.*

Now we are ready to define the non-three-colorable hypergraphs that we will realize with points and disks. For each m let $\mathcal{H}^2(m) = \mathcal{H}(T)$ where T is the m-ary tree of depth m. Clearly, $\mathcal{H}^2(m)$ is m-uniform and not two-colorable. We define non-three-colorable m-uniform hypergraphs $\mathcal{H}^3(m)$ based on them inductively.

First we create a sequence of hypergraphs. Let $\mathcal{F}_1(m) = \mathcal{G}_1$ and let v denote the single vertex of it. For for $i > 1$, let $\mathcal{F}_i(m)$ be the hypergraph that we get if we extend each edge of $\mathcal{F}_{i-1}(m)$ that contains v by $\mathcal{H}^2(m)$. Note that $\mathcal{F}_i(m)$ has only two types of edges. The first type are the ones that contain v, each of these contains exactly i vertices. (They are like the descendent edges.) The second type are those that were added in a copy of $\mathcal{H}^2(m)$, these contain exactly m vertices. (They are a bit like the sibling edges.) Therefore, $\mathcal{F}_m(m)$ is m-uniform. Also, by Claim 10 and induction, no $\mathcal{F}_i(m)$ is three-colorable.

Let $\mathcal{H}^3(m) = \mathcal{F}_m(m)$. For example, $\mathcal{F}_1(2)$ is $\mathcal{G}_1$ and $\mathcal{H}^2(2)$ is a triangle graph. Extending $\mathcal{G}_1$ through its single edge by a triangle gives us the complete graph on 4 vertices, so $\mathcal{H}^3(2)$ is just K_4.

3.1 Realizing $\mathcal{H}^3(m)$

Since $\mathcal{H}^3(m)$ was built by a series of extensions, it is enough to show that we can do each extension geometrically. This step is essentially the same as in [19].

Lemma 11 *Suppose a hypergraph $\mathcal{A}$ is realized with $(\mathcal{P}, \mathcal{D})$ such that every disk $D \in \mathcal{B} \subset \mathcal{D}$ has a point on its boundary that does not belong to the closure of any other disk $D' \in \mathcal{D}$. Then we can also realize $\mathcal{A} +_F \mathcal{H}(T)$ for any rooted tree T and any edge $F \in \mathcal{B}$, with a pair $(\mathcal{P}', \mathcal{D}')$, such that any disk $D \in \mathcal{B}$ and every new copy of F has a point on its boundary that does not belong to the closure of any other disk $D' \in \mathcal{D}'$.*

Proof. Suppose $D_F \in \mathcal{D}$ is the disk realizing the edge F. Then D_F has a point p on its boundary that does not belong to the closure of any other disk $D' \in \mathcal{D}$. Take a small circle C that is tangent to D at p. If we chose the radius of C to be small enough, then C will not intersect any of the disks. Now take $n = |V(T)|$ copies of D_F and rotate them slightly around the center of C. If all the

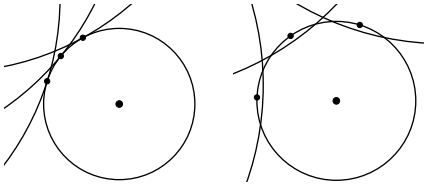


Figure 4: Extension

rotations are small, the resulting disks will be ε-close to D_F and by Observation 1 they will contain the same points as D_F. Also, if the angles of the rotations are different, then each new disk has a point on its boundary that does not belong to the closure of any other disk.

Then we enlarge each copy of D_F slightly, so that they intersect C but some part of their boundary remains uncovered by other disks. Place the points $q_1, \ldots, q_n$ in the intersections and use Theorem 8 to realize $\mathcal{H}(T)$. Since each disk in the realization of $\mathcal{H}(T)$ is close to C, they do not contain any point of $\mathcal{P}$. □

4 Stabbed unit disks

Here we prove Theorem 3, that given a fixed origin o, any point set $\mathcal{P}$ can be k-colored such that every unit disk that contains the origin and at least $8k-7$ points of $\mathcal{P}$ contains all k colors.

We call unit disks containing o *stabbed* unit disks. Take a representative disk for every $\mathcal{P}' \subset \mathcal{P}$ that can be obtained as the intersection of a stabbed unit disk and $\mathcal{P}$. Since $\mathcal{P}$ is finite, the collection of these representative disks, $\mathcal{D}$, is also finite. We divide the plane into four quarters by two perpendicular lines through o.

Lemma 12 *The boundaries of two stabbed unit disks intersect at most once in each quarter.*

(See Figure 5.)

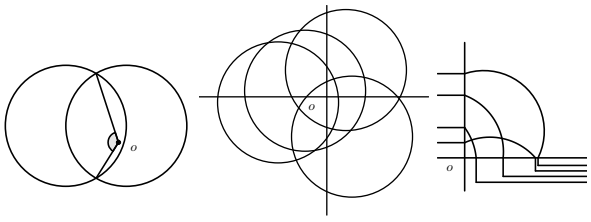


Figure 5: From stabbed disks to pseudolines.

Proof. Denote the two intersection points of the boundary of the two disks, D_1 and D_2, by x_1 and x_2. Since D_1 and D_2 have equal radii, the length of the arc they contain from each other is less than half of their perimeter. Thus, using Thales's theorem, for any inner point p of $D_1 \cap D_2$ the angle $x_1 p x_2$ is at least 90°. This proves the statement as $o \in D_1 \cap D_2$. □

This means that in each quarter the boundaries of the disks behave as pseudolines. Let us partition our point set $\mathcal{P}$ into four parts, $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4$, depending on which quarter the points are in. We will color each $\mathcal{P}_i$ separately.

Denote the quarter that contains the points of $\mathcal{P}_i$ by Q_i. Prolong the section of the boundaries falling into this quarter, i.e., $D \cap Q_i$ for each $D \in \mathcal{D}$, such that they do not intersect outside Q_i and all of them stretches from "left infinity" to "right infinity". Without loss of generality, we can suppose that o is above all these curves. The (upward) regions that are bounded by such curves form (upward) *pseudohalfplanes*. Generalizing a theorem of Smorodinsky and Yuditsky [22], the following polychromatic coloring result is known about them.

Theorem 13 (Keszegh-Pálvölgyi [10]) *Given a finite collection of points and pseudohalfplanes, the points can be k-colored such that every pseudohalfplane that contains at least $2k-1$ points will contain all k colors.*

We can apply this theorem to $\mathcal{P}_i$ (ignoring $\mathcal{P} \setminus \mathcal{P}_i$) and our regions. From this we get that any stabbed disk that contains at least $2k-1$ points of $\mathcal{P}$ from the quarter Q_i will contain all k colors. Repeating this for all four $\mathcal{P}_i$, we obtain by the pigeonhole principle that any stabbed disk containing at least $4(2k-2)+1 = 8k-7$ points of $\mathcal{P}$ contains all k colors.

For $k = 2$, this gives that 9 points per disk are enough, though this is probably far from being tight. From below, we could only find a construction showing that 3 points per disk is not enough.

5 Open questions

One of the most interesting questions left open is about unit disks. Is there an m such that every point set in the plane can be three-colored such that every unit disk containing exactly m points contains at least two colors?

In fact, this conjecture has a strengthening in the dual, cover-decomposition problem. Is there an m such that every m-fold covering of $\mathcal{P}$ by disks can be partitioned into three parts such that any two parts cover $\mathcal{P}$?

If there is a counterexample to these questions, then it has to be quite different from ours, as already the two-color version [17] needed the realization of a different hypergraph.

Acknowledgment

We would like to thank Balázs Keszegh for useful discussions and for reading the draft of this manuscript and also our anonymous reviewers for their valuable suggestions.

References

[1] Eyal Ackerman, Balázs Keszegh, and Dömötör Pálvölgyi. Coloring hypergraphs defined by stabbed pseudo-disks and *ABAB*-free hypergraphs. *Acta Math. Univ. Comenian. (N.S.)*, 88(3):363–370, 2019.

[2] Eyal Ackerman, Balázs Keszegh, and Máté Vizer. Coloring points with respect to squares. *Discrete & Computational Geometry*, 58(4):757–784, Dec 2017.

[3] Noga Alon, Guoli Ding, Bogdan Oporowski, and Dirk Vertigan. Partitioning into graphs with only small components. *J. Combin. Theory Ser. B*, 87(2):231–243, 2003.

[4] Andrei Asinowski, Jean Cardinal, Nathann Cohen, Sébastien Collette, Thomas Hackl, Michael Hoffmann, Kolja Knauer, Stefan Langerman, Michał Lasoń, Piotr Micek, Günter Rote, and Torsten Ueckerdt. Coloring hypergraphs induced by dynamic point sets and bottomless rectangles. In *Algorithms and data structures*, volume 8037 of *Lecture Notes in Comput. Sci.*, pages 73–84. Springer, Heidelberg, 2013.

[5] Jean Cardinal, Kolja Knauer, Piotr Micek, and Torsten Ueckerdt. Making triangles colorful. *J. Comput. Geom.*, 4(1):240–246, 2013.

[6] Louis Esperet and Gwenaël Joret. Colouring planar graphs with three colours and no large monochromatic components. *Combin. Probab. Comput.*, 23(4):551–570, 2014.

[7] Matt Gibson and Kasturi Varadarajan. Optimally decomposing coverings with translates of a convex polygon. *Discrete Comput. Geom.*, 46(2):313–333, 2011.

[8] Balázs Keszegh. Coloring half-planes and bottomless rectangles. *Computational geometry*, 45(9):495–507, 2012.

[9] Balázs Keszegh and Dömötör Pálvölgyi. Convex polygons are self-coverable. *Discrete Comput. Geom.*, 51(4):885–895, 2014.

[10] Balázs Keszegh and Dömötör Pálvölgyi. An abstract approach to polychromatic coloring: Shallow hitting sets in ABA-free hypergraphs and pseudohalfplanes. In Ernst W. Mayr, editor, *Graph-Theoretic Concepts in Computer Science - 41st International Workshop, WG 2015, Garching, Germany, June 17-19, 2015, Revised Papers*, volume 9224 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2015.

[11] Balázs Keszegh and Dömötör Pálvölgyi. Proper coloring of geometric hypergraphs. In *Symposium on Computational Geometry*, volume 77 of *LIPIcs*, pages 47:1–47:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[12] Jon M. Kleinberg, Rajeev Motwani, Prabhakar Raghavan, and Suresh Venkatasubramanian. Storage management for evolving databases. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 353–362, 1997.

[13] István Kovács. Indecomposable coverings with homothetic polygons. *Discrete & Computational Geometry*, 53(4):817–824, 2015.

[14] Nathan Linial, Jiří Matoušek, Or Sheffet, and Gábor Tardos. Graph coloring with no large monochromatic components. *Electronic Notes in Discrete Mathematics*, 29:115 – 122, 2007. European Conference on Combinatorics, Graph Theory and Applications.

[15] Sergey Norin, Alex Scott, Paul D. Seymour, and David R. Wood. Clustered colouring in minor-closed classes. *Combinatorica*, 39(6):1387–1412, 2019.

[16] János Pach. Covering the plane with convex polygons. *Discrete Comput. Geom.*, 1(1):73–81, December 1986.

[17] János Pach and Dömötör Pálvölgyi. Unsplittable coverings in the plane. *Advances in Mathematics*, 302:433–457, 2016.

[18] János Pach, Dömötör Pálvölgyi, and Géza Tóth. Survey on decomposition of multiple coverings. In *Geometry–Intuitive, Discrete, and Convex*, pages 219–257. Springer, 2013.

[19] János Pach, Gábor Tardos, and Géza Tóth. Indecomposable coverings. In *Discrete geometry, combinatorics and graph theory*, volume 4381 of *Lecture Notes in Comput. Sci.*, pages 135–148. Springer, Berlin, 2007.

[20] Dömötör Pálvölgyi. Indecomposable coverings with concave polygons. *Discrete Comput. Geom.*, 44(3):577–588, 2010.

[21] Dömötör Pálvölgyi and Géza Tóth. Convex polygons are cover-decomposable. *Discrete & Computational Geometry*, 43(3):483–496, Apr 2010.

[22] Shakhar Smorodinsky and Yelena Yuditsky. Polychromatic coloring for half-planes. *J. Combin. Theory Ser. A*, 119(1):146–154, 2012.

Appendix

Detailed description and proof of correctness of non-two-colorable construction

From properties (i) and (ii) we know that at the start of the k-th step every point in $Des(p_k)$ lies on the boundary of $B(p_k)$, and they do not belong to any disk in $\mathcal{D}_{Des}$. Suppose $S(p_k) = \{r_1, \ldots, r_l\}$. To maintain the three properties, we want to add the disks $B(r_1), \ldots, B(r_l)$, and by the end of the k-th step we want to place the points of $Des(r_i)$ on the boundary of $B(r_i)$.

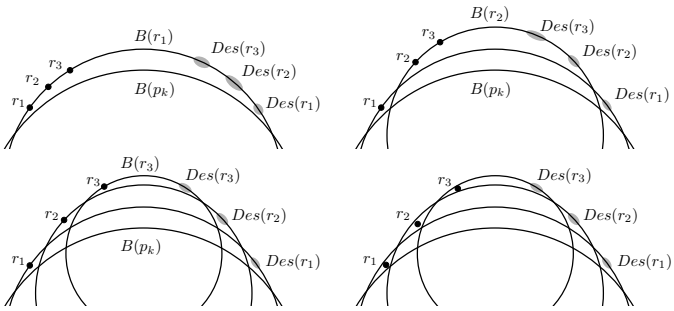


Figure 6: Figure 3 repeated.

To achieve this, we apply Observation 1 and Lemma 5 for each child in the following way. First apply Observation 1 for the points in $Des(p_k)$ and disks in $\mathcal{D}_{Des} \setminus \{B(p_k)\}$. Since the points in $Des(p_k)$ doesn't belong to the boundary of any other disk this is possible. We update δ to the result if it smaller than the current value.

Then we apply Lemma 5 for the boundary of the disk $B(p_k)$, the b_i-s are the points $\{r_1, \ldots, r_l\} \cup Des(r_l) \cup \cdots \cup Des(r_1)$ and ε is chosen to be the current value of δ. The points a and c have to be chosen carefully. We know that the points in $\{r_1, \ldots, r_l\} \cup Des(r_l) \cup \cdots \cup Des(r_1)$ lie on an arc of $B(p_k)$ that is not covered by any disk in $\mathcal{D}_{Des}$. We choose a and c on the two ends of this arc such that they are also not covered by any disk. Lemma 5 gives us a circle C', this defines $B(r_1)$, which is added to $\mathcal{D}_{Des}$. The position of the points in $\{r_1, \ldots, r_l\} \cup Des(r_l) \cup \cdots \cup Des(r_1)$ is updated according to the result of Lemma 5. As usual δ is also updated.

When we apply Observation 1 for the i-th time $(i > 1)$ we apply it for the points $\{r_i, \ldots, r_l\} \cup Des(r_l) \cup \cdots \cup Des(r_i)$ and disks in $\mathcal{D}_{Des} \setminus \{B(r_{i-1})\}$.

When we apply Lemma 5 for the i-th time $(i > 1)$, we apply it for the boundary of $B(r_{i-1})$, the b_i-s are the points in $\{r_i, \ldots, r_l\} \cup Des(r_l) \cup \cdots \cup Des(r_i)$ and ε is the current value of δ. The point a is chosen between r_{i-1} and r_i. The point c is chosen after the points of $\{r_i, \ldots, r_l\} \cup Des(r_l) \cup \cdots \cup Des(r_i)$ but before the points of $Des(r_{i-1}) \cup \cdots \cup D(r_1)$. If $Des(r_{i-1}) \cup \cdots \cup D(r_1)$ is empty, c is chosen before the arc of $B(r_{i-1})$ reaches any other disk. In the i-th case we get $B(r_i)$, which is added to $\mathcal{D}_{Des}$, and new positions for the points $\{r_i, \ldots, r_l\} \cup Des(r_l) \cup \cdots \cup Des(r_i)$ on $B(r_i)$. We update δ after each application of Lemma 5.

Finally, we finish the k-th step by moving $r_1, r_2, \ldots, r_l$ inside $B(r_1), B(r_2), \ldots, B(r_l)$ respectively, but at most δ far. Since r_i was lying on the boundary of $B(r_1)$, we can also ensure that they are not moved into any other disk. Then we add $r_1, \ldots, r_l$ to $\mathcal{P}_{fix}$. We update δ again by applying Observation 1 to $(\mathcal{P}_{fix}, \mathcal{D}_{Des})$.

Let us see why properties $(i), (ii), (iii)$ remain true during the run of the algorithm. Suppose they are true after the $(k-1)$-th step.

The first part of property (i) and property (ii) is maintained since we have created the disks $B(r_1), \ldots, B(r_l)$, and by the end of the k-th step the points of $Des(r_i)$ are on the boundary of $B(r_i)$. Points $(r_1, \ldots, r_l)$ were added to $\mathcal{P}_{fix}$.

The second part of property (i) could be violated in two ways. It could be that one of the new disks covers a point it should not. We have always chosen a and c such that this is avoided. The other possible violation is that we move a point into a disk. This is avoided, since if a vertex v lies on a disk D before moving, then we have updated δ for the disks in $\mathcal{D}_{Des} \setminus \{D\}$ right before moving the point. Also the movement is done by Lemma 5 so v cannot move into D.

As for property (iii), note that the only new disks in $\mathcal{D}_{Des}$ are $B(r_1), \ldots, B(r_l)$. Since (iii) was true before the step, $B(p_k)$ contains the points of $Q(p_k)$ and each of those are in $\mathcal{P}_{fix}$. When we add $B(r_1)$, it is δ-close to $B(p_k)$, so it contains exactly the points in $Q(p_k)$. Similarly, when $B(r_i)$ is added it is δ-close to $B(r_{i-1})$, so each of $B(r_1), \ldots, B(r_l)$ contains the points of $Q(p_k)$. When finally we move the points $r_1, r_2, \ldots, r_l$ inside $B(r_1), B(r_2), \ldots, B(r_l)$, respectively, we achieve that $B(r_i)$ contains the points of $Q(r_i)$.

We also need to check property (iii) for the disks that were already in $\mathcal{D}_{Des}$. Consider a disk $D \in \mathcal{D}_{Des}$. No point inside D was moved, since they are in $\mathcal{P}_{fix}$. The points in $\mathcal{P} \setminus \mathcal{P}_{fix}$ remain outside of D, since r_i only moves into one of the new disks and the rest of the points remain on the boundary of $\bigcup_{D \in \mathcal{D}_{Des}} D$. Hence property (iii) remains true.

Connection to clustering for planar graphs

We say that a graph G is c-colorable with clustering m if each vertex can be assigned one of c colors so that each monochromatic component has at most m vertices. For example, planar graphs are four-colorable with clustering 1 due to the Four Color Theorem. Clustering questions gained popularity recently, see [3, 6, 14, 15]. It has been shown [12] that for each m there exists a planar graph that is not three-colorable with clustering m. We reprove this result using Theorem 1.

Theorem 14 (Kleinberg et al. [12]) *For each $m > 0$ there exists a planar graph that is not three-colorable with clustering m.*

Proof. Take the point set $\mathcal{P}$ that realizes the non-three-colorable hypergraph $\mathcal{H}^3(m)$, and perturb the points so that no four lie on a circle. Consider the Delaunay graph of this point set, that is, we connect two points if they can be separated from the rest with a circle. From Theorem 1 we know that any three-coloring of this graph contains m monochromatic points that can be separated from the rest by a circle and it is well-known that m such points always form a connected component in the Delaunay graph, thus a monochromatic cluster in our case. □

Red-Blue Point Separation for Points on a Circle

Neeldhara Misra* Harshil Mittal† Aditi Sethia‡

Abstract

Given a set R of red points and a set B of blue points in the plane, the Red-Blue point separation problem asks if there are at most k lines that separate R from B, that is, each cell induced by the lines of the solution is either empty or monochromatic (containing points of only one color). A common variant of the problem is when the lines are required to be axis-parallel. The problem is known to be NP-complete for both scenarios [1, 10], and W[1]-hard parameterized by k in the former setting [5] and FPT in the latter [8]. We demonstrate a polynomial time algorithm for the special case when the points lie on a circle. Further, we also demonstrate the W-hardness of a related problem in the axis-parallel setting, where the question is if there are p horizontal and q vertical lines that separate R from B. The hardness here is shown in the parameter p.

1 Introduction

Given a set R of red points and a set B of blue points in the plane, the Red-Blue Separation (RBS) problem asks if there are at most k lines that separate R from B, that is, each cell induced by the lines of the solution is either empty or monochromatic (containing points of only one color). Equivalently, R is separated from B if, for every straight-line segment ℓ with one endpoint in R and the other one in B, there is at least one line in the solution that intersects ℓ. A common variant of the problem is when the solution lines are required to be axis-parallel (APRBS). Questions about the discrete geometry on red and blue points in general, and their separability using geometric objects in particular, are of fundamental interest. This makes RBS a well-studied problem on its own right. It is also motivated by the problem of univariate discretization of continuous variables in the context of machine learning [4, 9].

RBS is known to be NP-complete [10], APX-hard [1], and W[1]-hard when parameterized by the solution size [5]. The approximation hardness holds for the APRBS problem also, while in contrast the parameterized intractability is known only for the general RBS problem. Specifically, it is known that an algorithm running in time $f(k)n^{o(k/\log k)}$, for any computable function f, would disprove ETH [5]. This reduction crucially relies on selecting lines from a set with a large number of different slopes — in particular, the number of distinct slopes of the lines used is not bounded by a function of k.

For the case where $k = 1$ and $k = 2$, the problem is solvable in $O(n)$ and $O(n \log n)$ time respectively [7]. The problem is known to be FPT parameterized by the number of blue points (or the number of red points). A 2-approximation algorithm is also known for APRBS [1] by casting the separation problem as a special case of the rectangle stabbing problem[1]. We note that the 2-approximation algorithm and APX-hardness applies even to the separation of monochromatic point sets (where the goal is to separate *all* points from each other) and this problem is also known to admit a approximation (OPT log OPT)-approximation [6].

Our Contributions We first address a question raised in the discussions from [1]: *Do special cases admit better approximation ratios or even exact solutions?* We make partial progress on this question by answering it in the affirmative when the input points lie on a circle (which would be a special case[2] of points in convex position). In particular, we show that when points lie on a circle, both RBS and APRBS admit exact polynomial-time algorithms. Interestingly, the RBS problem is significantly simpler in this special case compared to its axis-parallel counterpart. For the latter, the size of the optimal solution is captured by a structural parameter of a graph that is naturally associated with the point set. Our proof is algorithmic and can be used to solve the associated computational question.

Further, we introduce a natural variant of APRBS, which is the (p, q)-APRBS problem. Here, as before, we are given a set of red and blue points, and the question is if there is a set of at most p horizontal lines and at most q vertical lines that separate R from B. We show that this problem is W[2]-hard when parameterized by p alone. Finally, we also show by a simple observation

*IIT Gandhinagar, neeldhara.m@iitgn.ac.in
†IIT Gandhinagar, mittal_harshil@iitgn.ac.in
‡IIT Gandhinagar, aditi.sethia@iitgn.ac.in

[1] This is based on the idea that lines separating R from B must stab all rectangles formed by red and blue points at the corners.

[2] We speculate that these ideas would also be relevant for the more general scenario of points in convex position. While the algorithm for RBS in fact works as-is for points in convex position, the details for the axis-parallel variant are less obvious.

that the $2^{O(|B|)}$ algorithm for APRBS from [5] can be improved to $2^{O(k \log |B|)}$.

The rest of the paper is organized as follows. We formally define the problems that we address in Section 2 and focus on the case of points on a circle in Section 3 for both RBS and APRBS. We describe the W[2]-hardness result for (p, q)-APRBS parameterized by p in Section 4. Due to lack of space, we make our remarks about the improved algorithm in the full version of the paper [11].

2 Preliminaries

For positive integers x, y, let $[x]$ be the set of integers between 1 and x, and $[x, y]$ the set of integers between x and y. Given a set of points $R \cup B$ in the plane, R is said to be separated from B by a collection of lines L if every straight-line segment with one endpoint in R and the other one in B is intersected by at least one line in L. We adopt the convention of requiring a "strict" separation, which is to say that no point in $R \cup B$ is on a separating line. We let $n := |R \cup B|$, $r = |R|$ and $b = |B|$. The computational problems that we study are the following.

Red-Blue Separation. (RBS) Given a set R of red points and a set B of blue points in the plane and a positive integer k as input, determine if there exists a set of at most k lines that separate R from B.

Axis-Parallel Red-Blue Separation. (APRBS) Given a set R of red points and a set B of blue points in the plane and a positive integer k as input, determine if there exists a set of at most k axis-parallel lines that separate R from B.

(p,q)-Axis-Parallel Red-Blue Separation. ((p,q)-APRBS) Given a set R of red points and a set B of blue points in the plane and positive integers p and q as input, determine if there exists a set of at most p horizontal and q vertical lines that separate R from B.

3 Points on a Circle

In this section, we focus on the special case when all the points lie on a circle C. Let $P = (R \cup B)$ denote a set of n points on a circle, with r red points and b blue points. As usual, R (respectively, B) denotes the set of red (respectively, blue) points. Without loss of generality, we assume that all points of P lie on an unit circle centered at the origin. Fix an order σ on $R \cup B$ based on the order of their appearance on the circle, starting at $(1, 0)$ and moving around the circle counterclockwise. We let r_i and b_j denote the i^{th} red and the j^{th} blue point that we encounter in this order. For a point p on the circle, we let $\mathsf{col}(p)$ denote the color of the point p.

We call a maximal sequence of monochromatic points in σ a *chunk*. Let $\mathcal{C}_P$ denote the set of chunks of P. In the order of their appearance on the circle, we denote the individual chunks by $C_1, \ldots, C_w$. The color of a chunk is the color of any point belonging to it. We refer to a chunk consisting of red (blue) points as a red (blue) chunk. We overload notation and let $\mathsf{col} : \mathcal{C}_P \to \{R, B\}$ be a function that returns the color of a chunk. The arc of C starting at the last point on the r^{th} chunk and the first point on the $(r+1)^{th}$ chunk is called a *switch*. Let $\mathcal{S}_P$ denote the set of switches of P. Note that any instance with w chunks has w switches. We denote the switches by $S_1, \ldots, S_w$ in the order of their appearance on the circle. Also note that w must always be even, and that there are $\frac{w}{2}$ red chunks and $\frac{w}{2}$ blue chunks.

We say that a switch S is *stabbed* by a line ℓ if $\ell \cap S \neq \emptyset$. We first make the following observation.

Proposition 3.1 *Let $P = (R \cup B)$ be a red-blue point set on a circle. If L is a set of lines that separates R from B, then every switch must be stabbed by some line in L.*

Proof. Suppose that there exist a switch $S_i \in \mathcal{S}_P$ that is not stabbed by any line from the set L. Note that S_i separates the chunks C_i and C_{i+1}. Without loss of generality, suppose $\mathrm{col}(C_i) = R$ and $\mathrm{col}(C_{i+1}) = B$. Since S_i is not stabbed by any line from L, the last point of C_i and first point of C_{i+1} are not separated, which leads to the contradiction of the fact that L separates R from B. Therefore, every switch must be stabbed by some line in L. □

Based on Proposition 3.1, we have that in an instance with w switches, $\frac{w}{2}$ is a lower bound on the optimum, since any line can stab at most two chunks. In the next subsection, we show that this can always be achieved by a set of general lines. For axis-parallel lines, we strengthen the lower bound further using an auxiliary graph structure on the switches, and demonstrate an algorithmic argument to match the stronger lower bound.

3.1 The Case of General Lines

With arbitrary lines, our strategy is simple: we "protect" each monochromatic chunk of a fixed color with a single line passing through it's adjacent switches. In particular, consider any chunk C_i such that $\mathsf{col}(C_i) = B$. Fix an arbitrary point p_i in the switch[3] S_{i-1} and another point q_i in the switch S_i. Let ℓ_i be the line passing through p_i and q_i and let $L := \{\ell_i \mid \mathsf{col}(C_i) = B\}$. In other words, L is the set of lines thus defined based on blue chunks. Note that there are $\frac{w}{2}$ lines in this solution, since we introduce one line for each blue chunk. Moreover, it is also easy to check that these lines sep-

[3] If $i = 1$, then we let $S_{-1} := S_w$.

arate R from B, since every blue chunk belongs to a separate cell of this configuration.

3.2 The Case of Axis-Parallel Lines

When we are restricted to axis-parallel lines in the solution, then the strategy described in the previous subsection would fail since the lines that are described need not be axis-parallel. A similar strategy does give us a simple 2-approximation, which we describe informally. Observe that each monochromatic chunk can be protected by a "wedge" consisting of a pair of axis-parallel lines. Indeed, consider the points p_i and q_i defined as before, and let T be the unique rectangle whose sides are axis-parallel and which has p_i and q_i as diagonally opposite corner points. Clearly, one of the other two corner points c lies inside C. We can now choose the two axis-parallel lines that contain the edges of the rectangle which intersect at c, and we have a wedge-like structure that protects the chunk (depending on the length of the chunk, note that the points of the chunk may be distributed over multiple cells). This gives us a solution with w lines, and is therefore a two-approximate solution.

We now demonstrate a stronger lower bound for the setting of axis-parallel lines. To this end, we introduce some terminology and define an auxiliary graph based on the point set P. We say that a pair of switches *face each other* if there exists a horizontal or vertical line that stab both of them. A switch which faces at least one other switch is said to be *nice*, a pair of switches facing each other is called a *nice pair*, and a switch that is not nice is said to be *isolated*. We define a graph based on P that has a vertex for every switch, and an edge between every pair of vertices corresponding to switches that are nice pairs. Formally, for a red-blue point set $P = (R \cup B)$ with w switches $S_1, \ldots, S_w$, we define the graph $G_P = (V_P, E_P)$ as follows: $V_P = \{v_j \mid 1 \leq j \leq w\}$ and $E_P = \{(v_i, v_j) \mid (S_i, S_j) \text{ is a nice pair}\}$.

Observe that every isolated switch of P corresponds to an isolated vertex of G_P. Recall that an *edge cover* of a graph G is a set of edges such that every vertex of the graph is incident to at least one edge of the set. Note that a minimum-sized edge cover can be found by greedily extending a maximum matching of a graph G. We use the abbrevation MEC to refer to a minimum edge cover. Let $I_P \subseteq V_P$ be the set of isolated vertices of G_P and let $H_P := G_P \setminus I_P$. We define $\kappa(G_P) := |I_P| + \mathsf{MEC}(H_P)$, where $\mathsf{MEC}(G)$ denotes the size of a minimum edge cover of the graph G. Our first claim is that any instance $P = (R \cup B)$ of APRBS requires at least $\kappa(G_P)$ lines to separate R from B. Next, we will show that this bound is tight.

Before stating the claims formally, we make some remarks about the bound. Note that this coincides with the bound obtained as a consequence of Proposition 3.1 when G_P has a perfect matching. Further, the bound is w when G_P is the empty graph, or equivalently, when every switch is an isolated switch. In this scenario, note that the approach described for the two-approximate solution will, in fact, yield an optimal solution. The intuition for the bound in the generic case is the association between lines and edges in a MEC: indeed, every edge e in G_P corresponds to a family of lines that stab switches corresponding to the endpoints of e. Our goal is to show that we can pick one line corresponding to each edge in the MEC and one line for each isolated switch in such a way that we separate R from B. However, it is easy to come up with examples where this does not happen, and indeed, the argument for the upper bound follows by making a bounded number of modifications to the set of lines that was proposed with guidance from the MEC. On the other hand, this association runs both ways, so we can recover subset of edges from any collection of lines separating R from B, using that stab two switches. If a solution uses fewer than $\kappa(G_P)$ lines, the hope is that the edges recovered lead us to an edge cover that has fewer edges than the MEC, which would be a contradiction. We now formalize both sides of this argument. We begin with the lower bound.

Lemma 3.1 *Let $P = (R \cup B)$ be a red-blue point set on a circle. Let L be a set of k axis-parallel lines that separate R from B. Then $k \geq \kappa(G_P)$.*

Proof. Consider any solution L that uses k axis-parallel lines. By Proposition 3.1, we know that every switch must be stabbed by some line from L. In particular, suppose there are α lines in L that stab a pair of (nice) switches, and β lines that stab one switch. Clearly, $\beta \geq |I_P|$, the number of isolated vertices in G_P.

Let X be the set of non-isolated vertices in G_P which are not covered by the edges corresponding to the α lines stabbing pairs of nice switches. Now, note that the switches corresponding to these non-isolated vertices in X must be stabbed by one of the β lines. So, $\beta \geq |X| + |I_P|$. Recall that $H_P = G_P \setminus I_P$ and $\mathsf{MEC}(H_P)$ covers every non-isolated vertex of G_P. Observe that $\mathsf{MEC}(H_P) \leq \alpha + |X|$, since the edges corresponding to the α lines stabbing pairs of switches can be extended by a collection of $|X|$ edges, one each for each non-isolated vertex that is not accounted for so far, to obtain a MEC for H_P. Adding both the inequalities above, we get:

$$\alpha + \beta + |X| \geq |X| + |I_P| + \mathsf{MEC}(H_P)$$
$$\Rightarrow \alpha + \beta \geq |I_P| + \mathsf{MEC}(H_P); \Rightarrow k \geq \kappa(G_P),$$

as desired. □

We now turn to the upper bound. In this section, we state some claims without proofs due to lack of space

and refer the reader to the full version of the paper for the detailed arguments.

Lemma 3.2 *Let $P = (R \cup B)$ be a red-blue point set on a circle. There exists a collection of at most $\kappa(G_P)$ lines that separate R from B.*

The proof of the upper bound is algorithmic, and we demonstrate it with a series of claims. To begin with, let $F_P \subseteq E_P$ be a MEC of H_P and let $t := |I_P|$. We define a set of lines L_0 as follows. For every edge $e = (v_i, v_j) \in F_P$, let ℓ_e be an arbitrary axis-parallel line passing through the switches S_i and S_j. For every isolated switch S_r, let ℓ_r be an arbitrary axis-parallel line stabbing S_r. Now define L_0 as the collection of all of these lines, i.e:

$$L_0 = \{\ell_e \mid e \in F_P\} \cup \{\ell_r \mid v_r \in I_P\}.$$

Note that $|L_0| = \kappa(G_P)$. If L_0 separates R from B, then we are done. Otherwise, we will obtain another set of axis-parallel lines that "dominates" L_0 in that it has the same size as L_0, separates all pairs of points separated by L_0 and at least one additional pair. To formalize this, we introduce the notion of a strictly dominating solution. For a set of lines L, let $\mathsf{sep}(L) \subseteq R \times B$ denote the set of red-blue point pairs that are separated by L. Given two collections of axis-parallel lines L and $L^\star$, we say that $L^\star$ *strictly* dominates L if $|L^\star| \leq |L|$ and $\mathsf{sep}(L) \subsetneq \mathsf{sep}(L^\star)$. We will now show that there exists a sequence of sets of axis-parallel lines $L_0, L_1, \ldots, L_g$ such that L_i strictly dominates L_{i-1} for all $1 \leq i \leq g$ and L_g separates R from B. Note that the number of steps is bounded by rb. Throughout, we will maintain the invariant that every switch is stabbed by at least one line. Note that this is true, in particular, for L_0.

Claim 3.1 *Every switch is stabbed by at least one line from L_0.*

For a collection of axis-parallel lines L, we say that a cell of L is *corrupt* if it is non-monochromatic, that is, if it contains at least one red point and at least one blue point. Note that L_0 contains at least one corrupt cell, otherwise we would be done. We consider all the possible ways in which a cell can intersect the circle underlying our point set.

Claim 3.2 *Let $\mathcal{R}$ be an axis-parallel rectangle and let C be a circle centered at the origin. Then $\mathcal{R} \cap C$ is either empty or consists of at most four disjoint arcs of C.*

Next, we note that any corrupt cell must contain at least two disjoint arcs of the circle.

Claim 3.3 *Let L be a set of lines that stabs every switch at least once, and let $\mathcal{R}$ be a corrupt cell of L. Then $\mathcal{R} \cap C$ contains at least two disjoint arcs of the circle C.*

We say that a cell $\mathcal{R}$ is *large* if $\mathcal{R} \cap C$ contains three or four disjoint arcs of C. We note that any instance can have at most one large cell.

We are now ready to describe the high-level strategy for obtaining a sequence of strictly dominating solutions. It turns out that if a corrupt cell consists of exactly two disjoint arcs, then depending on the "location" of the cell, there is a simple strategy that allows us to clean up the cell by flipping the orientation of one of the lines in the solution. In particular, and informally speaking, the strategy works for corrupt cells that are "above" ("below") the origin if all cells above it are monochromatic, or corrupt cells "to the left" ("to the right") the origin if all cells before (after) it are monochromatic. This gives us a natural sweeping strategy to clean up corrupt cells from four directions, while potentially getting stuck at a large cell "at the center". When the large cell is the only corrupt one, it turns out that there are a fixed number of configurations it can have when considered along with its surrounding cells, and for each of these cases, we suggest an explicit strategy to clean up the large cell to arrive at a solution with no corrupt cells at all. We now formalize this argument.

Let L denote the current solution: to begin with, L is L_0, and we describe a process to obtain a solution L' that strictly dominates L if L is not already a valid solution. Note that L divides the plane into vertical and horizontal strips, which we will refer to as the rows and columns of the solution. Also, we call a cell of this configuration *empty* if it does not contain any points of P. We first focus on corrupt cells that are *not* large. Consider a cell $\mathcal{R}$ whose intersection with C consists of exactly two disjoint arcs, say A_1 and A_2. Note that A_1 and A_2 lie in distinct quadrants. We call $\mathcal{R}$ a horizontal cell if these arcs lie in the first and second or the third and fourth quadrants; and we call $\mathcal{R}$ a vertical cell if these arcs lie in the first and fourth or the second and third quadrants. Note that the remaining possibilities do not arise with cells that are not large. We refer the reader to Figure 1 in the full version of the paper for a visual representation of these cases.

Consider the corrupt horizontal cell whose center has the largest y-coordinate in absolute value. This is either the top-most corrupt cell above the x-axis (Case 1) or the bottom-most corrupt cell below the x-axis (Case 2). If there are no corrupt horizontal cells, then consider the corrupt vertical cell whose center has the largest x-coordinate in absolute value. This is either the left-most corrupt cell to the left of the y-axis (Case 3) or the right-most corrupt cell to the right of the y-axis (Case 4).

Let us consider Case 1. Here, observe that any row above the row containing the cell $\mathcal{R}$ consists of at most

one non-empty cell and that all such cells are monochromatic by the choice of $\mathcal{R}$. Now, if the cell above $\mathcal{R}$ is monochromatic red and the arc in the first (second) quadrant consists of red points, then the top line of $\mathcal{R}$ can be flipped to a vertical line about the top-left (top-right) corner of the cell $\mathcal{R}$. It is easy to check that this solution strictly dominates L. The case when the cell above $\mathcal{R}$ is monochromatic blue can be argued similarly. We refer the reader to Figure 2 in the full version of the paper for an illustration of the switching strategies in these scenarios.

Case 2 is similar to Case 1 except that we argue relative to the cells below $\mathcal{R}$ rather than above it. In Cases 3 and 4, we flip vertical lines to a horizontal orientation, and the argument is based on monochromatic cells that lie to the left and right of $\mathcal{R}$, respectively. All the details are analogous to the case that we have discussed. Therefore, as long as the current solution has a corrupt cell that is not large, this discussion enables us to find a strictly dominating solution.

Now, the only case that remains is the situation where we have exactly one corrupt cell which is large. For a large cell we have four surrounding monochromatic or empty cells. The three or four arcs contained inside the large cell may also have red or blue points in different configurations. It turns out that each of these cases admits a new solution which makes all cells monochromatic. This can be established by inspection, and we refer the interested reader to the supplementary material that goes over all the individual cases[4]. Meanwhile, we refer the reader to Figure 3 in the full version of the paper for a stereotypical case and the corresponding strategy. Based on this discussion, we conclude with the formal statement of the main result of this section.

Theorem 3.1 *RBS and APRBS can be solved in polynomial time when the input points lie on a circle.*

4 W-hardness of (p,q)-Separation

In this section, we focus on the (p, q)-APRBS problem. Before describing our result, we briefly comment on the relationship between this problem parameterized by only the budget for horizontal lines (or vertical lines, by symmetry) and APRBS parameterized by the size of the entire solution. If APRBS had turned out to be W[1]-hard or W[2]-hard parameterized by k, then it would imply that (p, q)-APRBS is unlikely to be FPT parameterized by either p or q, since such an algorithm can be used as a black box to resolve the former question with only a polynomial overhead (guess p, q such that $p + q = k$). On the other hand, if (p, q)-APRBS turns out to be FPT parameterized by either p or q, then this would imply that APRBS is also FPT for the same reason. We show that (p, q)-APRBS is W[2]-hard when parameterized by p, the number of horizontal lines used in the solution. Therefore, our observation here establishes the hardness of the problem for a smaller parameter, and it does not have any direct implications for APRBS. Our result is also not implied by what is known about APRBS, since it turns out that the problem is FPT parameterized by k.

We reduce from the Colorful Red-Blue Dominating Set (C-RBDS) problem, which is defined as follows. The input is a bipartite graph $G = (R, B, E(G))$ along with a partition of R into k parts $R_1 \uplus \cdots \uplus R_k$. The question is if there exists a subset $S \subseteq R$ such that $|R_i \cap S| = 1$ for all $1 \leq i \leq k$ and that dominates every vertex in B; in other words, for all $v \in B$, there exists a $u \in S$ such that $(u, v) \in E(G)$. Such a set is called a colorful red-blue dominating set for the graph G. This problem is well-known to be W[2]-hard when parameterized by k [2]. Our reduction is inspired by the reduction from SAT used to show the hardness of the problem of separating n points from each other [1]. One aspect that is specific to our setting is ensuring that the budget for lines in one orientation is controlled as a function of the parameter.

Theorem 4.1 *(p, q)-APRBS is W[2]-hard when parameterized by p.*

Proof. Let $G = (R = R_1 \uplus \cdots \uplus R_k, B); k)$ be an instance of C-RBDS. Without loss of generality, we assume that every vertex $v \in B$ has the same degree d and that d is even[5]. We may also assume that all R_i's have the same number of vertices (padding R_i with $\max_{j=1}^{k}\{|R_j|\} - |R_i|$ dummy isolated vertices if required). We let $|R_1| = \cdots = |R_k| = m$ and $n := |B|$. We also assume that k is even, again without loss of generality. Finally, we impose an arbitrary but fixed ordering on each R_i and on the sets $N(v)$ (neighbours of v in G) for every $v \in B$.

It will be convenient to think of the point set of the reduced instance as lying within a sufficiently large bounding box, say $\mathcal{B}$. To describe the placement of the points, we impose an uniform $(k + 2) \times (n + 1)$ grid on $\mathcal{B}$, which divides $\mathcal{B}$ into $k + 2$ horizontal regions $H_0, H_1, \ldots, H_k, H_{k+1}$ (labeled from bottom to top) and $(n + 1)$ vertical regions $V_0, V_1, \ldots, V_n$ (labeled from left to right) which we call *tracks*. Each horizontal track H_i for $i \in [k]$ is divided further into $m + 2$ horizontal strips

[4] Please see the full version of this work for more details [11].

[5] When this is not the case, let $\Delta := \max_{v \in B}\{d(v)\}$. We may introduce an additional "dummy color" class R_0 with a forced dummy vertex (for example, by adding a d-star whose center is in B and whose leaves are in R_0), and for every vertex $v \in B$ we may introduce $\Delta - d(v)$ new pendant red neighbors of v in R_0. If d happens to be odd, use $\Delta + 1$ in this process instead of Δ to ensure that d is even.

and each vertical track V_j for $j \in [n]$ is divided further into $2d$ vertical strips. Within a horizontal track, the first and last horizontal strips are called *buffer zones*. Further, when we refer to the p^{th} horizontal strip within any horizontal track, the buffer zones are not counted. We refer the reader to Figure 4 in the full version of this paper for a visual representation of the reduced instance.

For $i \in [k]$, $j \in [n]$, $\alpha \in [m]$, and $\beta \in [2d]$, we refer to the intersection of the α^{th} horizonal strip in H_i and the β^{th} vertical strip in V_j as $Z_{ij}[\alpha, \beta]$. We note that two points that share the same x-coordinate (y-coordinate) have to be separated by a vertical (horizontal) line. We now describe three sets of points that we need to add: the first will lead us to a choice of a vertex from each R_i, the second set encodes the structure of the graph, and the third set ensures that the chosen set is indeed a dominating set by forcing the use of a budget in a certain way.

Selectors. Consider the first vertical track. Here, for any even (odd) $i \in [k]$, we add a red (blue) point to the top buffer zone and a blue (red) point to the bottom buffer zone of the i^{th} track. These $2k$ points are called the *selectors*. We ensure that all selectors have the same x-coordinate. Intuitively, the selector points ensure that any valid solution is required to use at least one horizontal line drawn in each of the k horizontal tracks — and the budget will eventually ensure that any valid solution uses exactly one. Which horizontal strip these lines end up in will act as a signal for our choice of vertices in the dominating set in the reverse direction.

Functional Points. Next, consider any vertex $v_j \in B$. For every $u \in N(B)$, we add a pair of red and blue points in $Z_{ij}[\alpha, 2\beta]$ if u is the α^{th} vertex of R_i and is the β^{th} neighbor of v_j. These points are added to the bottom-left and top-right corners of the box. If β is odd (even)[6], then the bottom-left corner gets a blue (red) point and the opposite corner gets the red (blue) point. These pairs of points will be referred to as the *functional pairs*. The functional pairs encode the structure of the graph, and we would like to ensure that the responsibility of separating at least one functional pair in each vertical track falls on a horizontal line used to separate the selectors. We force this by choosing an appropriately small budget for vertical lines, which ensures that not all separations can be accounted for using vertical lines. However, we still need to control how the vertical budget is utilized across different tracks. To this end, we introduce a special gadget that forces the use of a certain number of vertical lines in each vertical track.

[6]The organization of colors based on the parity of the columns in the case of functional pairs and rows in the case of selectors is to ensure that there are no additional separation requirements other than the ones that we desire to encode.

Guards. In the horizontal track H_0, we place d points, all with the same choice of y-coordinate which is arbitrary but fixed. Within the j^{th} vertical track, x-coordinate of the s^{th} point is chosen so that the point lies in the middle of the $(2s)^{th}$ vertical strip of V_j. The color of the first vertex in the track V_i is blue if i is odd and red if i is even. This ensures that for $2 \leq i \leq n$, the first point in the i^{th} track has the same color as the last point of the $(i-1)^{th}$ track. The colors of the remaining points are chosen so that consecutive points within the same track have distinct colors. Equivalently, the s^{th} guard vertex in the i^{th} track is blue (red) if s and i are both odd (even), and is red (blue) if s is odd (even) and i is even (odd). We refer to these points as *guards*.

We briefly discuss the role of the guard vertices: we note that the guards can be separated from each other by $(d-1)$ vertical lines, and since all guards have the same y-coordinate, this is the only way to separate them. However, there is no set of $(d-1)$ lines that can separate all the guards *and* all the associated functional pairs in any vertical track. The budget for the vertical lines will be such that we can only afford to separate the guards as we are required to do, and we will be forced to separate at least one functional pair using a horizontal line, which will essentially ensure that the selectors have chosen vertices corresponding to a dominating set.

We let $p := k + 2$ and $q := (d-1)n + 1$. This mostly completes the description of the construction. Due to lack of space, we defer the argument for equivalence and some minor details in the construction to the full version of the paper [11]. □

5 Concluding Remarks

We showed that RBS and APRBS are polynomial-time solvable when points lie on a circle. Further, we introduced a natural variant that separates out the budget for horizontal and vertical lines in the axis-parallel variant, and demonstrated that (p, q)-APRBS is W[2]-hard when parameterized by p. We expect a natural adaptation of these arguments to work for points in convex position as well. In the general setting, since APRBS is FPT when parameterized by k [8], the question of whether the problem admits a polynomial kernel would be natural to explore further. Our W[1]-hardness reduction for (p, q)-APRBS may provide some starting points towards an answer in the negative — in its present form the parameter k of the reduced instance depends on k, d, and n. APRBS would not admit a polynomial kernel (under standard complexity-theoretic assumptions) if this dependence can be reduced to k and n only [3].

References

[1] G. Călinescu, A. Dumitrescu, H. J. Karloff, and P. Wan. Separating points by axis-parallel lines. *International Journal of Computational Geometry and Applications*, 15(6):575–590, 2005.

[2] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 1st edition, 2015.

[3] M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and ids. *ACM Transactions Algorithms*, 11(2):13:1–13:20, 2014.

[4] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence.*, pages 1022–1029. Morgan Kaufmann, 1993.

[5] P. Giannopoulos, E. Bonnet, and M. Lampis. On the parameterized complexity of red-blue points separation. *Journal of Computational Geometry*, 10(1):181–206, 2019.

[6] S. Har-Peled and M. Jones. On separating points by lines. *Discret. Comput. Geom.*, 63(3):705–730, 2020.

[7] 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.

[8] S. Kratsch, T. Masarík, I. Muzi, M. Pilipczuk, and M. Sorge. Optimal discretization is fixed-parameter tractable. *CoRR*, abs/2003.02475, 2020.

[9] J. Kujala and T. Elomaa. Improved algorithms for univariate discretization of continuous features. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Database (PKDD)*, volume 4702 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2007.

[10] N. Megiddo. On the complexity of polyhedral separability. *Discrete Computational Geometry*, 3:325–337, 1988.

[11] N. Misra, H. Mittal, and A. Sethia. Red-blue point separation for points on a circle, arXiv/2005.06046, 2020.

A lower bound on the number of colours needed to nicely colour a sphere

Péter Ágoston*

Abstract

The Hadwiger–Nelson problem is about determining the chromatic number of the plane (CNP), defined as the minimum number of colours needed to colour the plane so that no two points of distance 1 have the same colour. In this paper we investigate a related problem for spheres and we use a few natural restrictions on the colouring. Thomassen showed that with these restrictions, the chromatic number of all manifolds satisfying certain properties (including the plane and all spheres with a large enough radius) is at least 7. We prove that with these restrictions, the chromatic number of any sphere with a large enough radius is at least 8. This also gives a new lower bound for the minimum colours needed for colouring the 3-dimensional space with the same restrictions.

1 Introduction

1.1 Colourings of the plane

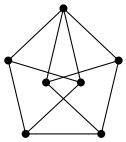


Figure 1

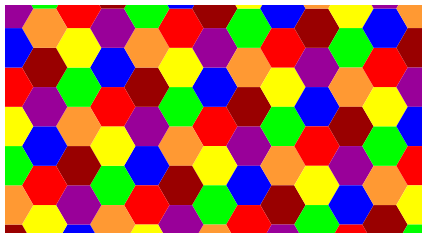


Figure 2

The Hadwiger–Nelson problem is a well-known problem in combinatorial geometry. It asks to determine the chromatic number of the plane (CNP), i.e., the minimum number of colours needed to colour the plane so that no two points of distance 1 have the same colour. Alternatively, it is the chromatic number of the graph of unit distances on the plane. Since 1950 it has been known that $4 \leq CNP \leq 7$. The lower bound was obtained by Nelson (1950), but it can be most easily proven by using a graph called the Moser spindle (Figure 1) (Moser, Moser (1961) [M]), while the upper bound was given by Isbell (1950), using the colouring in Figure 2. Since 2018, it is also known that $CNP \geq 5$ (de Grey [dG]).

The problem has some variations:

If we restrict the colour classes to be measurable, the best known lower bound for the number of colours needed is also 5 (Falconer (1981) [F]) and the best known upper bound is also 7 (also from Figure 2).

If we restrict the colour classes to be the unions of shapes bounded by Jordan curves (such a shape is called a tile and such a colouring is called a tile-based colouring or simply a tiling), then the best known lower bound for the number of colours needed is 6 (Townsend (2005) [T]) and the best known upper bound is also 7.

Thomassen also defined a type of tiling:

A colouring of a surface with a metric is *nice*, if it is a tiling, all tiles have diameter less than 1, all pairs of tiles with the same colour have distance more than 1 and all tiles are simply connected. We refer to such a colouring as a nice tiling.

He also proved the following theorem:

Theorem 1 *[T] Suppose a surface S satisfies the following three conditions for some natural number k:*

1. Every noncontractible simple closed curve has diameter at least 2.

2. If C is a simple closed curve of diameter less than 2, then the area of $int(C)$ is at most k.

3. The diameter of S is at least $12k + 30$.

Then every nice tiling contains at least 7 colours.

Since the plane satisfies the conditions, the theorem proves that every nice tiling of the plane contains at least 7 colours.

Note that the statement for the plane also follows from a relatively easy proof using Lemma 2 and the fact that a triangulated planar graph has $3n - 6$ edges.

1.2 Colouring of spheres

We can define the chromatic number of a sphere of radius r similarly to the planar case: it is the minimum number of colours needed to colour the points of a sphere of radius r such that no two points with Euclidean distance 1 have the same colour.

Much less is known of the value of this number compared to the planar case.

It is known that the chromatic number of a sphere of radius r is at least 4 if $r \geq \frac{1}{\sqrt{3}}$. For $r > \frac{\sqrt{3}}{2}$ Moser's

*MTA-ELTE Lendület Combinatorial Geometry (CoGe) Research Group, Eötvös Loránd University, Budapest, Hungary, supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017- 00002). agostonp@cs.elte.hu

spindle gives the lower bound, for smaller values, a generalized version of Moser's spindle is used. (Simmons (1976) [S])

It is also known that the chromatic number of any sphere is at most 15, even with all of the above defined restrictions, as the 3-dimensional space has a 15-tiling (Radoičić, Tóth (2003)[RT]), which can be used to generate such a colouring.

Recently, a 7-colouring of large enough spheres have also been found by Tom Sirgedas, as part of the Polymath 16 project.[1]

Also, the minimal number of colours needed for a nice tiling of a large enough sphere is at least 7, which follows from Theorem 1.

The main result of this paper is improving this number to 8.

Note that some sources mentioned earlier that the chromatic number of all spheres is at most 7 [BMP] [HDCG]. It seems (from personal communication through Dömötör Pálvölgyi) that the authors expected that a colouring similar to that of Isbell in Figure 2 also works for spheres. The present paper disproves this assumption, though it does not contradict to the chromatic number of spheres being at most 7.

This result also improves the lower bound for the minimal number of colours needed for a nice tiling of the 3-dimensional space, for which problem the best known bound was 6 for the general colouring case (Nechushtan (2002) [N]).

2 Results

2.1 Preliminary statements

Definition 1 *We call a colouring of a graph nice, if there are no two different vertices within distance at most* 2*, which are coloured with the same colour. Alternatively, a nice colouring can be defined as a colouring of G, which is also a proper colouring for G^2 (the square of G).*

Lemma 2 *If a tiling of a surface is nice, then applying the same colouring to the adjacency graph of the tiles also gives a nice colouring.*

The proof is in the Appendix.

Lemma 3 *If S is a sphere with radius $r \geq \frac{2}{\pi}$ and $A \subseteq S$ such that the spherical diameter of A is less than* 1*, then there is a unique connected component of $S \setminus A$, which contains all points of S with the exception of at most an open disk of radius* 1*.*

The proof is in the Appendix.

[1] https://groups.google.com/forum/#!topic/hadwiger-nelson-problem/tSOs7MypGxE

Lemma 4 *If S is a sphere with radius larger than $\frac{2}{\pi}$ and the adjacent tiles $A, B \subseteq S$ both have spherical diameter less than* 1*, then there is a unique connected component of $S \setminus (A \cup B)$, which contains all points of S with the exception of at most an open disk of radius* 1*.*

The proof is in the Appendix.

For any set $A \subseteq S$ or two sets $A, B \subseteq S$, call the unique component described above the large component of $S \setminus A$ or $S \setminus (A \cup B)$, respectively, and all the other components the small components.

2.2 The main result

Theorem 5 *There is no nice tiling with at most* 7 *colours of a large enough (radius $r \geq 18$) sphere S, even if we generalize the definition and allow tiles not to be simply connected.*

In order to make the proof more legible, we give an outline of the main steps.

Suppose that there exists such a colouring of S and take the adjacency graph G of the tiles such that all tiles are represented by one of their points. First, by deleting some vertices and edges, we get rid of all multiple edges and cut vertices and get a graph G', which is a triangulated planar graph and still has the property that all of its pairs of neighbouring vertices have distance less than 2. If we had a nice tiling of S, we also have a colouring of this graph, which is not only nice, but also no two vertices with distance at most 1 get the same colour. This will lead to a contradiction.

Since G' is a triangulated planar graph with maximum degree at most 6 (otherwise its colouring could not be nice), it has at most 12 vertices with degree less than 6 (exactly 12 if counted with multiplicity given by the differences of 6 and the degrees of these vertices). These vertices are called irregular vertices.

Also, for some subsets of G' that only have vertices with degree 6, there exists a function to an infinite triangular grid such that the mapping is a local isomorphism in all vertices.

Now we have three cases.

The first case is when all of the irregular vertices are close to each other. In this case, we find a cycle c_1 of bounded length separating them from most of S. And from the mapping we can get from the latter part to the triangular grid, we can prove that this part has a bounded graph size. So we can get to a contradiction by finding a vertex far away from the irregular vertices, which exists if r is large enough.

In the second case the irregular vertices can be divided into two groups both of cardinality 6 (counted with multiplicity), where the elements of the two groups have a large enough distance from each other, while inside one group, the distances are bounded. In this case,

we can construct two cycles c_2 and c_3 of bounded length separating these two groups, which are close enough to the first and the second group, respectively. Then again we get a contradiction from the mapping of the part between the two cycles to the triangular grid: we find a cycle in this part that goes through two nearly antipodal points, but its graph length is not larger than $max(l(c_2), l(c_3))$.

Finally, the last case is when there is at least one way to divide the irregular vertices into two groups such that no two points from different groups are close to each other and the cardinality of the two groups (counted with multiplicity) is not divisible by 6. In this case, we get a contradiction by examining the exact way to colour parts of the infinite triangular grid.

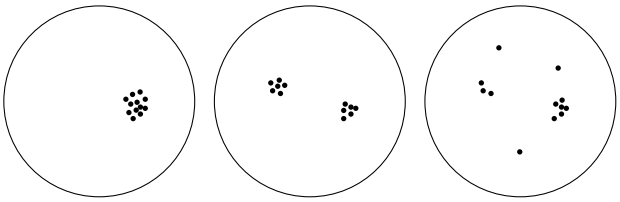


Figure 3: Case 1, Case 2 and Case 3

Now we continue with a detailed proof.

Proof. For the sake of simplicity, first we will use spherical distances in the calculations (so we are now solving the problem when tiles have spherical diameter at most 1 and tiles of the same colour have spherical distance more than 1) and we will only convert the problem to the Euclidean distance definition in the end. So now suppose that S has a radius $r \geq 17.9$.

Suppose we have a nice tiling of S.

Take the graph G where the tiles $T_1, ...T_n$ are represented by vertices $v_1, ...v_n$ and the neighbouring ones are connected. The colouring of the tiles gives a nice colouring of this graph by Lemma 2, which we will use in the proof.

We can also get rid of vertices or edges, and prove the statement for the remaining subgraph as it also implies that the original graph cannot be coloured with a nice colouring with 7 or less colours either. First, eliminate the multiple edges (a multiple edge can occur if two tiles have a border made out of disjoint segments): if we find a pair of parallel edges between v_i and v_j, we can just merge them and delete everything between them (between meaning the vertex set corresponding to the small components of $S \setminus (T_i \cup T_j)$). We also eliminate cut vertices (these correspond to not simply connected tiles) by deleting all vertices corresponding to the tiles in the small components of $S \setminus v_i$ for any cut vertex v_i. Also, we can eliminate those complete graphs with more than 3 vertices that represent points where more than 3 tiles meet: we just take an arbitrary triangulation of them as if the tiles would not exactly meet in one point. This way we have got a triangulated planar graph G', and from now on, triangles will always mean the empty 3-cycles of G'. If G' has n' vertices, it has $3n' - 6$ edges, which means that the sum of the degrees of its vertices is $6n' - 12$. And since there are no vertices with neighbours of the same colour (the colouring for G' is also nice), all vertices have degree at most 6. So there are at most 12 vertices having degree less than 6. Call them irregular vertices and for any irregular vertex, let its multiplicity be the difference of its degree from 6. From the above, there are exactly 12 irregular vertices, if we count them with multiplicity. Also, call the set of irregular vertices I and call the elements of $V(G') \setminus I$ regular vertices.

Now draw G' on S so that $v_i \in T_i$ and the edges are represented by simple Jordan curves satisfying the following conditions:

1) The two endpoints of the image of an edge e has the two vertices e is incident to as its endpoints.

2) If the border of two tiles T_i and T_j contains more than one point, then draw the edge between v_i and v_j so that it only contains points from these two tiles. This also means that all points of the edge have distance less than 2 from both v_i and v_j.

3) For any point in which more than three tiles meet, the edges corresponding to pairs of tiles which only border each other in this point only run through border segments starting from the meeting point. Also, the edges should run so close to the common border point of the tiles that all points of all edges have distance less than 2 (measured on S) from both of the endpoints of that particular edge.

Lemma 6 *Any (open or closed) disk D on S with radius at least 1 contains at least one vertex from G'.*

The proof is in the Appendix.

So it is enough to prove the following (stronger) version of Theorem 5:

Suppose we have a sphere S with radius $r \geq 17.9$ and a fully triangulated planar graph G' on the surface of S, which has n' vertices, all of its vertices have distance less than 2 on S, all open unit disks on S contain at least one vertex and all of the points of all of its edges have distance less than 2 from both of its respective endpoints. Then G' cannot be coloured with 7 colours in a way so that any two vertices of the same colour have graph distance at least 3 in G'.

Now continue with some definitions:

For any two subsets of S, let their *spherical distance* be their spherical distance on S (denoted by $dist_S(a, b)$).

For any two subgraphs G_a and G_b of G', let their *graph distance* mean the smallest graph distance in G' occuring between their vertices (denoted by $dist_{G'}(a, b)$).

For any 3-cycle c in G' that has a side, which is

Let the *graph length* of a path or closed path p in G' be the number of its edges. We denote it by $l(p)$.

Let the *spherical broken line* of a path or closed path p in G' be the curve defined by connecting neighbouring vertices of p with spherical segments instead of the edges connecting them.

Let the *broken line length* of a path or a closed path p in G' be the length of the spherical broken line belonging to p. We denote it by $L(p)$.

Let the *i-neighbourhood* of a vertex v of G' be the subgraph of G' induced by those vertices, which have graph distance at most i from v (denoted by $N_i(v)$).

Let the *strict i-neighbourhood* of a vertex v of G' be the subgraph of G' induced by those vertices, which have graph distance exactly i from v (denoted by $n_i(v)$).

For a vertex v and a set S of vertices, let $e(v, S)$ mean the number of edges starting from v and ending in any of the vertices of S.

We will use the following lemmas later in the proof:

Lemma 7 *For any path or closed path p in G', $L(p) < 2l(p)$.*

Proof. All edges connect points in neighbouring tiles meaning that they have a distance less than 2 as both of the endpoints have a distance at most 1 from an arbitrarily chosen border point. And by summing these inequalities, we get the statement. □

Lemma 8 *If we take a subgraph G^* of G', which does not include any irregular vertices and is defined by a simply connected region S^* on S such that those vertices are included, which are inside S^* and those edges are included, which are fully inside S^*, then there exists a function φ from the vertices and edges of G^* to an infinite triangular grid T (for the sake of simplicity, suppose that it is made up of regular triangles) fulfilling the following criteria:*

1) It keeps the incidence relation between vertices and edges.

2) For any vertex $v \in G^*$, if for two edges e_1 and e_2 in G^*, that have v as an endpoint, there are exactly k edges of G' between them going around v in a positive order, then there are exactly k edges of T between $\varphi(e_1)$ and $\varphi(e_2)$ going in a positive order around $\varphi(v)$.

The above colouring is unique up to isometries preserving orientation.

The proof is in the Appendix.

Lemma 9 *We can find a similar φ function if G^* is defined by a subset S' of S that is homeomorphic to $S^1 \times [0, 1]$ (like S minus two disjoint disks) and still does not contain irregular vertices and we also require G^* to be connected, but here the codomain of φ will be the set of (possibly infinite) sets of vertices in case of vertices and the set of (possibly infinite) sets of edges for edges. Here we require from φ that for any vertex $v \in G^*$, and an edge $e \in G^*$ incident with v all of the elements of $\varphi(v)$ are incident with at least one element of $\varphi(e)$. We also require that for any vertex $v \in H$ and any element $v' \in \varphi(v)$, we can choose an element from all the φ's of the edges incident to v such that they are all incident to v' and for any two of them, they have exactly as many edges between them going around v' in a positive order as the corresponding edges in G^* have going around v in a positive order.*

The proof is in the Appendix.

Analogously to the colouring of the plane by Isbell, call a colouring of the vertices of the infinite triangular grid T an *Isbell colouring* if it is constructed in the following way:

We take a vertex in the grid and colour it and its neighbours with 7 different colours. We then tile the grid with the disjoint translates of this coloured hexagon.

Such a colouring is trivially nice and periodical, thus any Isbell colouring of T can be generated using any of the vertices of T as the starting vertex. Also, for all colourings of the starting hexagon, there are exactly two ways to colour T depending on how we place the hexagons compared to each other. Also, all Isbell colourings can be generated with the above procedure starting from any hexagon formed by a vertex and its 6 neighbours.

Lemma 10 *The graph in Figure 4 can only be nicely 7-coloured by a part of an Isbell colouring.*

The proof is in the Appendix.

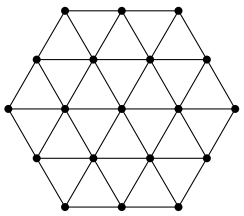


Figure 4

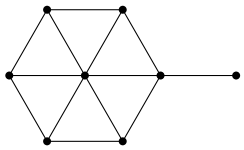


Figure 5

Lemma 11 *If we embed the graph in Figure 5 in the infinite triangular grid, then any colouring of it is contained in at most one Isbell colouring of T.*

Proof. The hexagon part determines the colouring of that particular hexagon, while the remaining vertex leaves at most one of the two colourings that can be generated from that particular colouring of the hexagon. □

Definition 2 *If we have a cycle c in G' then let one of its sides be called as the inside and the set of vertices of G' in it be called V_i, while the set of edges in it be called E_i, while the other one being the outside and call the set of vertices of G' in it V_o and the set of edges in it E_o. The curvature of c is $\sum_{v \in c} 2 - e(v, V_i)$.*

(Note that if c does not contain irregular vertices, then this definition is trivially equivalent with $\sum_{v \in c} e(v, c \cup V_o) - 4$.)

Lemma 12 *The curvature of a cycle c is equal to 6 minus the number of irregular vertices in the inside of c (counted with multiplicity).*

The proof is in the Appendix.

Let H be the graph with vertex set I and edges connecting the pairs of vertices, which have graph distance at most 3 in G'.

Now take a connected component H_i of H and take a spanning tree of H_i. For any edge e of this spanning tree find a path of length at most 3 in G' connecting the two endpoints of e (per definition, such a path exists) and take the union of these paths, which is a graph in G'. Take a spanning tree of this graph and call it H'_i. Do this for all components of H and call the union of these trees H'.

Lemma 13 *The H'_i's have graph distance at least 2 from each other in G'.*

The proof is in the Appendix.

Lemma 14 *If the number of vertices in H_i (counted with multiplicity) is n_i, then $|V(H'_i)| \leq 3n_i - 2$, $|E(H'_i)| = |V(H'_i)| - 1 \leq 3n_i - 3$ and $\{v \in G' | dist_{G'}(v, H_i) = 1\} \leq 5n_i$*

The proof is in the Appendix.

Now take an H_i and take the union U_i of the triangles (borders included) that have at least one vertex in common with H_i.

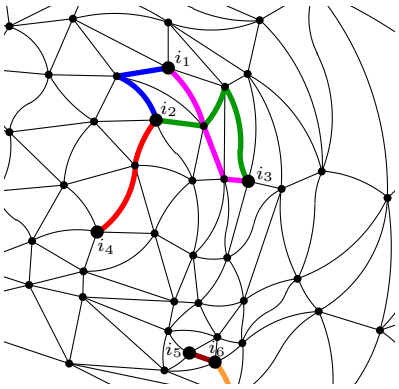


Figure 6: A part of G' with the chosen paths connecting the vertices of I highlighted

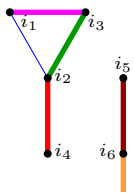


Figure 7: The part of H corresponding to this part of G'

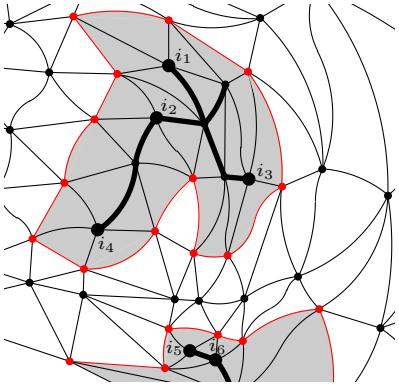


Figure 8: The corresponding part of H' denoted by bold, the U_i's denoted by grey and the c'_is denoted by *red*.

Lemma 15 *There exists a point $O_i \in S$ for which all the vertices of H'_i fit into a disk D_i of radius $3n_i - 3$ around O_i, all the vertices belonging to U_i fit into a disk D'_i of radius $3n-1$ around O_i and all the edges belonging to U_i fit into a disk D''_i of radius $3n - 3$ around O_i.*

The proof is is in the Appendix.

Let c_i be the cycle that borders the connected component of $S \setminus U_i$ that contains $S \setminus D''_i$. c_1 is trivially formed by vertices having graph distance 1 from H'_i meaning that it has at most $5n_i$ vertices because of Lemma 14. Also, per definition, it is inside D''_i.

Now we have three cases:

Case 1: H is connected.

Let H_1 be the only component of H. Then $l(c_1) \leq 60$ as of Lemma 14 and its vertices fit into an open unit disk D'_1 of radius 35 around O_1, while its edges fit into an open unit disk D''_1 of radius 37 around O_1.

Lemma 16 *For any vertex v of $G' \cap S_1$, its graph distance from c_1 is at most 10.*

The proof is in the Appendix.

But there is a vertex of G' inside the open unit disk around the antipodal of O_1, which has distance more than $r\pi - 36 > 20$ from D'_1. And since all of this disk is outside D''_1 (the distance of O_1 and its antipodal is at least $\pi \cdot 17.9 > 56 > 37 + 1$), the vertex inside it is a vertex of S_1, so it should have graph distance at most 10 and thus, spherical distance less than 20 from all the vertices of c_1, which is a contradiction.

Case 2:

H has two connected components and both have vertex number 6 (counted with multiplicity). Call these components H_2 and H_3

Lemma 15 and the subsequent statement yield that there exists a cycle c_2 of graph length at most 30 separating H'_2 from all the vertices outside a disk D''_2 of radius 19. Similarly there exists a cycle $c_3$of graph length

at most 30 separating H_3' from all the vertices outside a disk D_3'' of radius 19. Now define the interior of c_2 ($int(c_2)$) as the component of $S \setminus c_2$ containing H_2' and the interior of c_3 ($int(c_3)$) as the component of $S \setminus c_3$ containing H_3'. Since all the vertices of c_2 and c_3 have graph distance 1 from H_2' and H_3', respectively, from Lemma 13 neither H_2' has a common vertex with c_3, neither H_3' has a common vertex with c_2. So neither of c_2 or c_3 has vertices both in the interior and the exterior (the opposite component to interior) of the other one. Also, it is not possible that their interiors are covering S completely as they both fit into an open disk of radius 19 and S cannot be covered by two disks of this size. So we have two possibilities:

The interior of one of c_2 and c_3 is completely inside the interior of the other one. In this case, we can apply the argument used in Case 1 as the one having the other one in its interior also has all the vertices from I inside it, has length at most $30 < 60$ and fits into a disk of radius $19 < 35 < 37$.

In this case, define G_2 as the graph $((ext(c_2) \cup c_2) \cap (ext(c_3) \cup c_3)) \cap G'$ (where $ext(c_2)$ and $ext(c_3)$ denote the exteriors of c_2 and c_3 (the opposite side as their interiors)).

(A third possibility would be that $ext(c_2) \subseteq int(c_3)$ and $ext(c_3) \subseteq int(c_2)$, but it is clearly impossible due to their sizes.)

Now from Lemma 9 we can find a φ function running from G_2 to the infinite triangular grid T.

Now we will use the following lemma:

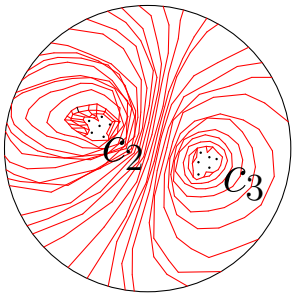


Figure 9

Lemma 17 *There exists a series of cycles ($\Gamma_0, ..., \Gamma_k$ for some k) in G' with $\Gamma_0 = c_2$ and $\Gamma_k = c_3$ satisfying 3 conditions:*

1) All of them have graph length at most 30, and thus, broken line length less than 60.

2) For any i, j with $|i - j| = 1$ and any vertex v of Γ_i, there is a vertex of Γ_j neighbouring v in G' and thus, having spherical distance less than 2 from it.

3) For any i, j with $|i - j| = 1$ and any edge e of Γ_i, there is a vertex of Γ_j having graph distance at most 1 from both of the endpoints of e in G', and thus, having spherical distance less than 4 from all of the points of e.

The proof is in the Appendix.

Now we can finish the proof for Case 2 using the following lemma:

Lemma 18 *At least one of $\Gamma_0, ..., \Gamma_k$ goes through two points that have spherical distance at least $r\pi - 5$.*

The proof is in the Appendix.

And since all of these curves have graph length at most 30, from Lemma 7 they also have broken line length less than 60. But from Lemma 18, there is one with broken line length more than $2 \cdot (r\pi - 5) \geq 2 \cdot (17.9\pi - 5) > 102$, which is a contradiction.

Case 3: Neither of the conditions of the previous cases hold.

Lemma 19 *If Case 3 holds, then there exists a cycle c_4 in G', all of whose vertices have graph distance at least 2 from all the irregular vertices and which separates them into two groups so that both of the groups has a cardinality not divisible by 6 (counted with multiplicity).*

The proof is in the Appendix.

Lemma 20 *For all vertices of c_4, a spanning subgraph of the 2-neighbourhood can be obtained as the image of an incidence and orientation preserving function Ψ from the hexagonal graph in Figure 4.*

The proof is in the Appendix.

Lemma 21 *The curvature of c_4 is divisible by 6.*

The proof is in the Appendix.

And this gives us a contradiction as the curvature of c_4 is not divisible by 6 according to Lemma 12

So it is impossible to have a nice tiling of a sphere if we are using spherical distances and $r \geq 17.9$. From this, it is impossible to have a nice tiling of a sphere if we are using Euclidean distances and $r \geq 18$ as if we have a sphere of radius $r \geq 18$ with a nice tiling, then a scaled version of the tiling would give a nice tiling for a sphere of radius $\frac{1}{2 \arcsin \frac{1}{2r}} \geq 17.9$ using the spherical distance version, which is a contradiction. □

3 Acknowledgement

I would like to thank my supervisor Dömötör Pálvölgyi for suggesting the problem, for his help in preparing this manuscript and all his guidance throughout the last few years. I would also like to thank Balázs Keszegh for his professional support.

References

[BMP] P. Brass, W. O. J. Moser, J. Pach: Research Problems in Discrete Geometry (2005)

[C] D. Coulson: On the chromatic number of plane tilings, *J. Aust. Math. Soc.* 77 (2004), 191–196.

[dG] A. D. N. J. de Grey: The chromatic number of the plane is at least 5, *Geombinatorics* (2018), 28: 18–31.

[F] K. J. Falconer: The Realization of distances in measurable subsets covering $\mathbb{R}^n$, J. Combin. Theory Ser. A 31 (1981) 184–189.

[HDCG] Handbook of Discrete and Computational Geometry, Third Edition J.E. Goodman, J. O'Rourke and C. D. Tóth, editors, CRC Press LLC, Boca Raton, FL, 2017. ISBN 978-1498711395 (68 chapters, xix + 1928 pages).

[M] L. Moser, W. Moser: Solution to Problem 10, *Can. Math. Bull.* **4** (1961)

[N] O. Nechushtan: On the space chromatic number, *Discrete Mathematics* **256** (2002), 499–507.

[RT] R. Radoičić, G. Tóth: Note on the Chromatic Number of the Space, *Discrete and Computational Geometry. Algorithms and Combinatorics* **25** (2003) 695–698., a preprint can be found at `http://www.cs.bme.hu/~geza/chromatic.pdf`

[S] G.J. Simmons: The chromatic number of the sphere. *J. Austral. Math. Soc. Ser. A,21:473–480, 1976.*

[T] S. P. Townsend: Colouring the plane with no monochrome unit, *Geombinatorics* **XIV(4)** (2005), 181–193.

[Th] C. Thomassen: On the Nelson Unit Distance Coloring Problem, *Amer. Math. Monthly* **106** (1999) 850–853.

3.1 Appendix

Proof of Lemma 2

The statement not being true would mean that there is a nice tiling for some surface, for which two vertices corresponding to tiles of the same colour have graph distance 1 or 2, but in the first case, their distance on the surface would be 0, while in the second, it would be less than 1, as the diameter of their common neighbour is less than 1. So the colouring would not be nice, so the statement is true.

Proof of Lemma 3

If we take an open disk D of radius 1 around any point of A, it covers A, so $S \setminus D \subseteq S \setminus A$ and since $S \setminus D$ is connected, all of its points are in the same connected component of $S \setminus A$. And this component is unique as $S \setminus D$ is a closed disk of radius $r\pi - 1$, so none of the other components satisfy the property described in the statement of the lemma.

Proof of Lemma 4

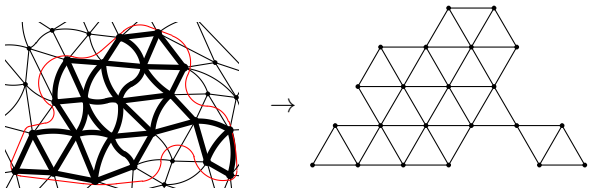
If we take an open disk D of radius 1 around a common border point of A and B, it covers both A and B, so $S \setminus D \subseteq S \setminus (A \cup B)$ and since $S \setminus D$ is connected, all of its points are in the same connected component of $S \setminus (A \cup B)$. And this component is unique as $S \setminus D$ is a closed disk of radius $r\pi - 1 > 1$, so none of the other components satisfy the property described in the statement of the lemma.

Proof of Lemma 6

If the center O of D belongs to a tile that is represented in G', then the vertex representing it has distance less than 1 from it, so it is inside D.

If O belongs to a tile that is not represented in G' because it is in the small component of $S \setminus t$ for some tile t that is represented in G' or in the small component of $S \setminus (t_1 \cup t_2)$ for some tiles t_1 and t_2 represented in G' (it is possible that O is also in the small component for some tile or tiles that are not represented in G'), then we can find a segment going through O that has both of its endpoints in the same tile represented in G' or at least on its borders and which has length less than 1. So it has distance less than 1 from the vertex representing this tile, which is thus in D.

Proof of Lemma 8



The statement can be proven by induction for the number of vertices of G^*:

If G^* does not have any vertices, the statement is trivial. Now suppose that it has m vertices and for all smaller vertex numbers, we have already proven the statement.

Suppose G^* is connected, otherwise the statement is trivial (its connected components can be defined by connected subsets of S, from which we can apply the induction hypothesis).

There is at least one (in fact, at least 4) irregular vertex in G', and it is not contained in G^*, thus if G^* is not empty, there is a vertex v inside G^*, which has at least one neighbour in $G' \setminus G^*$.

If v is a cut vertex in G^*, then examine the parts it separates G^* to (v included). All such graphs also can

be determined by a simply connected subset of S and they all have a smaller number of vertices than G^* had, and since we already have φ's for these smaller graphs, by choosing $\varphi(G^*)$ arbitrarily and rotating the images of the components around it appropriately, we get an appropriate φ.

If v is not a cut vertex, then suppose there are two edges e and e' of $G' \setminus G^*$ starting from v. Then we can find points $P \in e \setminus S^*$ and $P' \in e' \setminus S^*$ that means that the curve starting from P and ending in P' going through v on e and e' separates S^* into at least two separate parts. Thus, since v is not a cut vertex, at most one of the above parts contains any vertices of G^* meaning that the edges inside G^* that are starting in v form a connected interval among all the edges in G' starting from v. And because G' being triangulated, they form a chain in which all the neighbouring pairs of edges have 1 edge between them around their common vertex in the appropriate orientation. So the φ function belonging to $G^* \setminus v$ translates them to the subset of a regular hexagon around one vertex in T, so $\varphi(v)$ can be placed in its center, and the new φ we get (by also translating the edges from v into appropriate places) satisfies the conditions.

If G^* is connected, then φ is unique with respect to isometry as starting from a vertex and deciding which direction will be which, we always can continue in only one way.

Proof of Lemma 9

Take a simply connected covering space of S'.

Again, we can suppose that we only care about connected subgraphs.

We can find a function φ' from the pre-image of G^* (call this pre-image $G^{*\prime}$) to T in the same way we did it above, despite $G^{*\prime}$ containing an infinite number of vertices and edges: we simply use induction by always expanding $G^{*\prime}$, but never deleting anything from it and also we always keep the graph connected. And since the φ' of these

And from this we get the φ function mentioned in the statement: in any vertex or edge it will take the set of the φ''s of the pre-images of the particular vertex or edge.

And this φ will satisfy the conditions of the lemma as a consequence of the definition of covering spaces.

Proof of Lemma 10

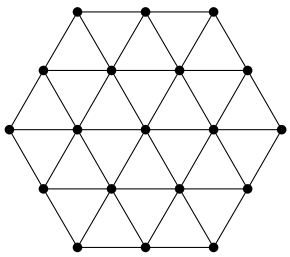


Figure 10

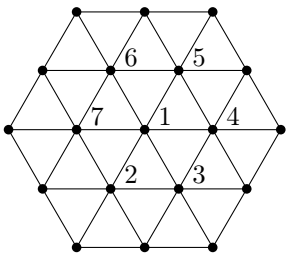


Figure 11

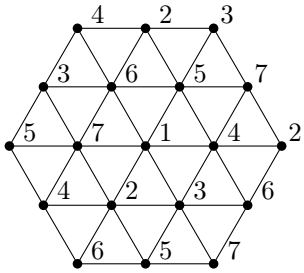


Figure 12

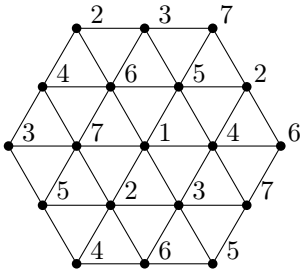


Figure 13

Colour the central vertex and its neighbours first (Figure 11). We now have to colour the remaining 12 vertices so that all border vertices of the central hexagon get exactly one neighbour from all colours (except for its own colour). And since for all colours from 2 to 7, there are exactly 3 coloured vertices that lack a neighbour with that colour and all of the uncoloured vertices border 1 or 2 of the coloured ones, we must use all six of these colours at least twice. But since there are 12 uncoloured vertices in total, we must use all of them exactly twice. From the above conditions, there are only two possibilities for choosing the vertices with colour 2 and from here, all the other colours follow (see Figure 12 and Figure 13).

Proof of Lemma 12

We will contract c into one triangle from the triangulation in the inside of c using the following steps:

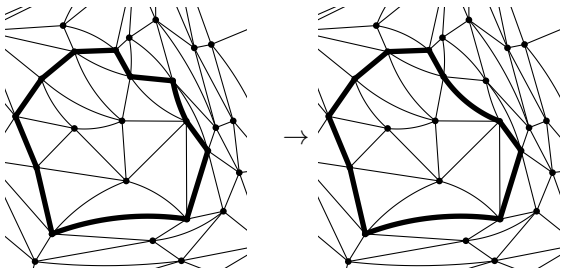


Figure 14: Step type 1

1. If we find an edge in E_i that connects two vertices v_a and v_b which are both on c and have distance 2 in

c, then we throw out the two edges between v_a and v_b and replace them with this edge and we also reduce E_i accordingly (as seen in Figure 14).

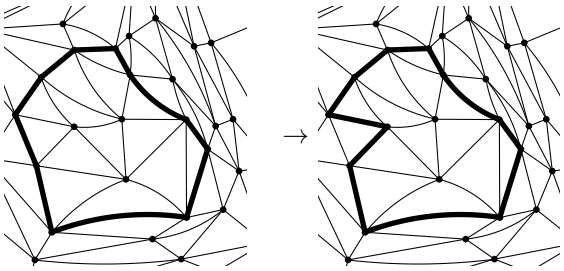


Figure 15: Step type 2

2. If we find a vertex v_c in V_i that is neighbouring two vertices v_d and v_e in c that are neighbouring each other in c, then we delete the edge v_dv_e from c and replace it with the edges v_dv_c and v_cv_e and then reduce V_i and E_i accordingly (as seen in Figure 15).

First I will prove that with always performing one of the steps above until it becomes impossible, the procedure we get is finite and ends in c being a triangle with its empty side being the inside:

If we are only looking at the subgraph of $c \cup E_i$ which is spanned by the vertices of c, it contains at most $n-3$ diagonals, so there is a vertex v_f (actually, at least 3) of c with no diagonals starting from it. And then if c is not already a triangle, the edge v_f forms with one of its neighbours (v_g) either belongs to a triangle that connects v_g with the other neighbour of v_f or to a triangle that connects both v_f and v_g with a vertex from V_i. In the first case, we can perform step 1, while in the second case, we can perform step 2. (And if c is a triangle with its empty side being the inside, then we cannot perform any of the steps.) And the number of triangles inside c always decreases with exactly 1, so the procedure is always finite.

If we perform step 1, the curvature of c does not change as the two summands belonging to v_a and v_b increase by 1, while the summand $2-0=2$ belonging to the vertex we have thrown out gets out from the sum.

If we perform step 2 and v_c is a regular vertex, the curvature of c does not change either, since the two summands belonging to v_d and v_e increase by 1 each and the new summand belonging to v_c is $2-4=-2$.

If we perform step 2 and v_c is an irregular vertex with multiplicity k, the curvature of c increases with k, since the two summands belonging to v_d and v_e increase by 1 each and the new summand belonging to v_c is $2-4+k = k-2$.

So since in the beginning all irregular vertices were in V_i and in the end no vertices remained in V_i and the curvature increased with k exactly if we deleted an irregular vertex of multiplicity k from V_i, otherwise it remained unchanged, the curvature have increased with the number of irregular vertices originally in V_i (counted with multiplicity). Also, the curvature of the triangle (with the empty side being the inside) is trivially 6, and this finishes the proof.

Proof of Lemma 13

Per definition, $dist(v, V(H_i)) \leq 1$ for all $v \in V(H_i')$. So if two vertices of some distinct graphs H_i' and H_j' have distance at most 1, then there also exist two vertices of H_i and H_j with distance at most 3, which contradicts to H_i and H_j being separate components of H.

Proof of Lemma 14

Since H_i has n_i vertices counted with multiplicity, it has at most n_i vertices counted without multiplicity. And we have drawn $|V(H_i)|-1 \leq n_i - 1$ paths all of length at most 3 (and thus all having at most 2 interior vertices) between them, thus, their union has at most $3n_i - 2$ vertices. And since H_i' is a subgraph of this union, it also has at most $3n_i-2$ vertices. And since it is a tree, $|E(H_i')| = |V(H_i')| - 1 \leq 3n_i - 3$.

So only the third statement remains to be proved. H_i' contains n_i irregular vertices (counted with multiplicity), so $\sum_{v \in H_i'} deg_{G'}(v) = 6 \cdot |V(H_i')| - n_i$. And from these, $|\{(v,e) | v \in V(H_i'), e \in E(G' \setminus H_i'), v \in e\}| = \sum_{v \in H_i'} deg_{G'}(v) - 2 \cdot |E(H_i')| = 4 \cdot |V(H_i')| - n_i + 2$. And all edges of H_i' border exactly two triangles and all triangles are bordered by at most two edges of H_i' since H_i' is a tree. Denote the number of triangles with one side in H_i' by k_1 and the number of triangles with two sides in H_i' by k_2. From the above, we know that $k_1 + 2k_2 = 2 \cdot |E(H_i')| = 2 \cdot |V(H_i')| - 2$. The number of edges of $G' \setminus H_i'$ connecting two vertices of H_i' is at least k_2, since any triangle with two sides in H_i' has such an edge as its third edge, and such an edge cannot belong to two different such triangles as then there would be a 4-cycle in H_i'. Thus, the number of edges of G' having one endpoint in H_i', while the other one in $G' \setminus H_i'$ is at most $|\{(v,e) | v \in V(H_i'), e \in E(G' \setminus H_i'), v \in e\}| - 2k_2 = 4 \cdot |V(H_i')| - n_i + 2 - 2k_2$ as all such edges are in $G' \setminus H_i'$ and all edges of $G' \setminus H_i'$ connecting two vertices from H_i' were counted twice in the above calculation. Now take a vertex v for which $dist_{G'}(v, H_i') = 1$ and take the set τ of triangles which have a side in H_i' and their third vertex is v. It is trivial that every such triangle is bordered by exactly two edges connecting v to H_i' and all edges are contained as an edge of at most two such triangles. Also, the latter inequality cannot be strict in case this particular edge is also bordering a triangle that is not in τ. So at least one of the following possibilities hold:

1) There are more than $|\tau|$ (so at least $|\tau|+1$) edges connecting v with H_i'

2) No edges connecting v with H_i' border the union of the triangles from τ.

But 2) is only possible if τ is empty (in which case,

1) still holds as there is at least one edge connecting v with H'_i) or if τ contains all triangles bordering v in which case the cycle formed by the neighbours of v is a subgraph of H'_i, which is a contradiction as H'_i is a tree. So 1) holds.

And by summing up such inequalities for all v's having graph distance 1 from H'_i, we get that $|\{v|v \in V(G'), dist_{G'}(v, H'_i) = 1\}| \leq |\{(v,w)|(v,w) \in E(G'), v \in V(H'_i), w \in V(G \setminus H'_i)\}| - k_1 \leq 4 \cdot |V(H'_i)| - n_i + 2 - 2k_2 - k_1 = 2 \cdot |V(H'_i)| + 4 - n_i \leq 5n_i$.

Proof of Lemma 15

Since $|V(H'_i)| \leq 3n_i - 2$, if H' is a centered tree, there is a vertex with graph distance at most $1.5n_i - 1$ from all of its vertices, while if it is a bicentered tree, there is an edge, whose endpoints both have graph distance at most 16 from all of its vertices. In the former case, the vertices of H' fit into an open disk of radius 32 because of Lemma 7, while in the latter case, the vertices of H' fit into an open disk of radius 33 because of Lemma 7. So in both cases the vertices of H' fit into a disk D_1 of radius 33 centered around a point on S called O_1. And since the vertices of U have graph distance at most 1 from H'_i, they

Proof of Lemma 16

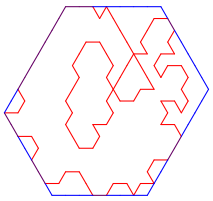


Figure 16

Take the cycle c'_1 in T represented by the blue cycle in Figure 16, where the red one represents $\varphi(c_1)$ (and the purple segments are their common edges). If we name the length of the ith side of c'_1 A_i for $1 \leq i \leq 6$ and we choose a vertex p_i of $\varphi(c_1)$ on all 6 of them, then for all i ($i = 1, 2, 3$) the two parts of $\varphi(c_1)$ separated by p_i and p_{i+3} both have at least $|A_{i+1}| + |A_{i+2}|$ segments parallel with A_{i+1} or A_{i+2} (i counted modulo 3), so $\varphi(c_1)$ has at least $2|A_{i+1}| + 2|A_{i+2}|$ such segments in total. And from summing up the three inequalities we get this way, if we combine it with $|A_i| + |A_{i+1}| = |A_{i+3}| + |A_{i+4}|$, we get $l(c_1) = l(\varphi(c_1)) \geq \sum_{i=1}^{6} |A_i| = l(c'_1)$ (the first equality is trivial).

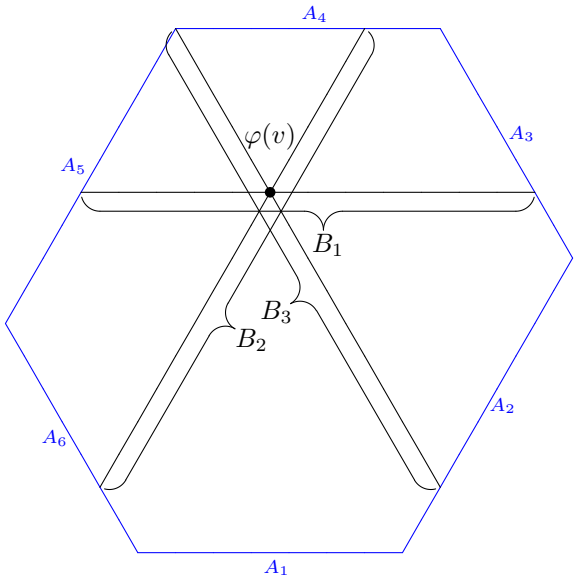


Figure 17

Now take a vertex v in G_1. If we call the segments going through $\varphi(v)$ parallel with A_1, A_2 and A_3 and ending in c'_1 B_1, B_2 and B_3, respectively (as in Figure 17), then we can write the following inequalities:

$|B_i| \leq |A_i| + \frac{|A_{i-1}|}{2} + \frac{|A_{i+1}|}{2}$ (for $i = 1, 2, 3$ if i is counted mod 6)

$|B_i| \leq |A_{i+3}| + \frac{|A_{i+2}|}{2} + \frac{|A_{i+4}|}{2}$ (for $i = 1, 2, 3$ if i is counted mod 6)

Summing up these 6 inequalities, we get $\sum_{i=1}^{3} 2|B_i| \leq \sum_{i=1}^{6} 2|A_i|$ and if we combine this with the fact that the distance of $\varphi(v)$ from c'_1 is at most $\frac{\min(|B_1|,|B_2|,|B_3|)}{2}$ we get $dist_{G'}(\varphi(v), c'_1) \leq \left\lfloor \frac{l(c'_1)}{6} \right\rfloor \leq \left\lfloor \frac{l(c_1)}{6} \right\rfloor \leq 10$ (where $dist_{G'}(\varphi(v), c'_1)$ denotes the graph distance of $\varphi(v)$ and c'_1 in G'.

Now if we define P_{min} as a path of minimal length from $\varphi(v)$ to c'_1, we can start a path from v so that the images of its vertices by φ are the vertices of P_{min} in the same order. And we can always take a step such that the image of the next vertex will be the next vertex in P_{min} until we reach c_1. And this can happen the latest when the φ of the path reaches c'_1, so $dist_{G'}(v, c_1) \leq dist_{G'}(\varphi(v), c'_1) \leq 10$.

Proof of Lemma 17

We will first contract G_2 into a cycle using the following six kinds of steps:

(The figures below represent the $\varphi(G_2)$ and $\varphi(c_2)$, which means that red broken line ($\varphi(c_2)$) occasionally can seemingly cross itself. From the curvature of both c_2 and c_3 being 0, it is easy to see that the sum of the angles of the turns is 0 for both $\varphi(c_2)$ and $\varphi(c_3)$, thus for any vertex or edge from G_2, its images by φ are periodical translates of each other.)

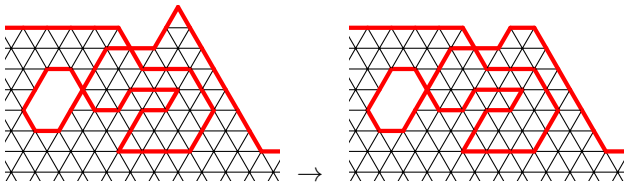


Figure 18: A vertex with degree 2 in G_2 is removed from G_2 and c_2 is changed accordingly.

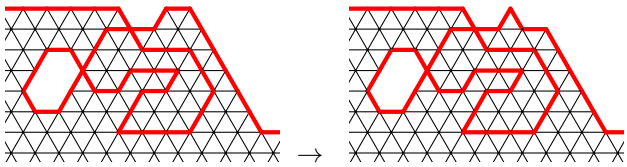


Figure 19: A vertex with degree 3 in G_2 is removed from G_2 and c_2 is changed accordingly.

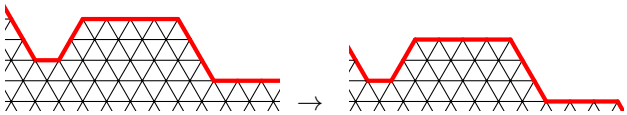


Figure 20: c_2 is shifted in some direction towards c_3 without adding anything to G_2.

The other three possibilities are doing the same kinds of steps with c_3.

Now we will prove that with such steps, we always can contract G_2 into a cycle without adding any vertex to G_2 in any step.

First, suppose that c_2 and c_3 have no common edges. Then if any of $\varphi(c_2)$ and $\varphi(c_3)$ is not a line, we can find a convex turn in one of them (a vertex of c_2 or c_3 in which $deg_{G_2}(c_2) \leq 3$ or $deg_{G_2}(c_3) \leq 3$, respectively) because both of them has a curvature of 6. So one of the first two kind of steps (or their c_3 counterparts) can be applied.

Now suppose that c_2 and c_3 have no common edges, but both of them are a line. In this case, we can apply Step type 3 without adding any vertex to G_2.

Now suppose that there is at least one edge in which c_2 and c_3 meet. In this case there is at least one subset $\hat{S}$ of S between c_2 and c_3 that is simply connected and is fully bordered by c_2 and c_3. And if we count $\hat{S}$ as the interior, its border has a curvature of 6, so even if in the two vertices, where c_2 and c_3 meet, the border takes a sharp convex turn (one, in which the degree towards $\hat{S}$ is 2, it still has to take a convex turn somewhere elsewhere, in which case, we can leave this particular vertex from c_2 or c_3.

Thus, the only case we cannot take such a step is if c_2 and c_3 coincide. And since the number of triangles between c_2 and c_3 always decreases with at least one, in finitely many steps, the procedure ends in G_2 being a cycle.

Now take every step in which c_2 was changed and name the original c_2 $c_2^{(0)}$, c_2 after the first step changing it $c_2^{(1)}$, c_2 after the second such step $c_2^{(2)}$, ... until we get to $c_2^{(p)}$ (where p is the number of steps changing c_2). Similarly, if the number of steps changing c_3 is q, we can define cycles $c_3^{(0)}$, $c_3^{(1)}$, ..., $c_3^{(q)}$.

Now take $k = p + q$ and define $\Gamma_0 = c_2^{(0)}$, $\Gamma_1 = c_2^{(1)}$, ..., $\Gamma_p = c_2^{(p)} = c_3^{(q)}$, $\Gamma_{p+1} = c_3^{(q-1)}$, ..., $\Gamma_k = p_3^{(0)}$.

For these cycles, the first condition of the lemma trivially applies, since all of the above steps decrease the graph length for both c_2 and c_3, so since originally, c_2 and c_3 did not have graph length more than 30, none of the Γ_i's ($i = 0, 1, ..., k$) have. So their broken line length is not more than 60.

The second and the third condition also can easily checked for both the $c_2^{(i)}$ ($i = 0, 1, ..., p$) and the $c_3^{(i)}$ ($i = 0, 1, ..., q$)

Proof of Lemma 18

First, for all $i = 0, ..., k$ let Γ_i' be the antipodal curve of Γ_i. Now define $int(\Gamma_i)$ as the connected component of $S \setminus \Gamma_i$, which includes the vertices of I_2, and let the other connected component be called $ext(\Gamma_i)$. Now define $int(\Gamma_i')$ and $ext(\Gamma_i')$ as the antipodal sets of $int(\Gamma_i)$ and $ext(\Gamma_i)$, respectively. Now define a function $f(i) = dist_S(V(\Gamma_i), ext(\Gamma_i')) - dist_S(V(\Gamma_i), int(\Gamma_i'))$. The first half is 0 if and only if at least one of the vertices of Γ_i is in $ext(\Gamma_i') \cup (\Gamma_i')$, while the second half is 0 if and only if at least one of the vertices of Γ_i is in $int(\Gamma_i') \cup (\Gamma_i')$. Thus, at least one of the two halfs is always zero and the sign of $f(i)$ is determined by which one is not zero. And since $V(\Gamma_0) \subseteq D_2''$ and $int(\Gamma_0')$ is inside the antipodal of D_2'', they are disjoint and their distance is positive. Similarly, $V(\Gamma_k)$ and $ext(\Gamma_k)$ are disjoint and their distance is positive. Thus, $f(0) < 0$ and $f(k) > 0$. Now take the first i, for which $f(i)$ is positive. If $f(i-1) \leq -3$, that means that all the vertices of Γ_{i-1} have a distance at least 3 from $int(\Gamma_{i-1}')$, so the vertices of Γ_i are further from $int(\Gamma_{i-1}')$ than 1, since all of them has distance less than 2 from at least one of the vertices of Γ_{i-1}. But also, for all border points of $int(\Gamma_{i-1})$, there exists a point of $ext(\Gamma_i)$ with distance less than 4 from it, so $f(i) = dist_S(V(\Gamma_i), ext(\Gamma_i)) < 4 - 1 = 3$. Thus, at least $|f(i-1)| < 3$ or $|f(i)| < 3$ is true, so one of the two cycles includes a vertex and another point, whose antipodals have spherical distance 3. And since the latter is 2 away from a vertex of the same cycle, we have found two vertices on one of the cycles with distance at least $r\pi - 5$ from each other.

Proof of Lemma 19

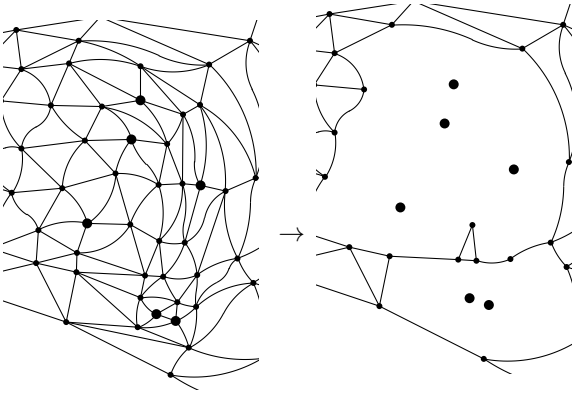


Figure 21: A part of G

Figure 22: The same part of G_3

Let the components of H be H_4, H_5, ... Now take the graph G_3 which we get from G' by deleting all the vertices that have graph distance at most 1 (with respect to G') from any of the points of I and all the edges that are incident to these vertices. All the irregular vertices, which are in different connected components of H are in different connected components of $S \setminus G_3$, since if there would be a path on S connecting two irregular vertices in different components of H, the triangles, edges and vertices it meets would all have graph distance at most 1 from some irregular vertex, which gives us a contradiction.

Now take an H_a which does not have a vertex number divisible by 6 (counted by multiplicity). If the point set G_3 contains a simple closed curve that separates the vertices of H_a from all the other irregular vertices, then this curve only can be a cycle in G_3 and it is applicable for c_4. If G_3 does not contain such a curve, it only can mean that the connected component of $S \setminus G_3$ belonging to I_i (call it C_i) separates the rest of S into at least two parts of which at least two contains a positive number of irregular vertices. Then we can separate C_i from both of these two parts by a cycle c_a and c_b, and it is trivial that both of these cycles separate the irregular vertices into two non-empty sets and at least one of the cycles divides I unevenly, so it can be chosen for c_4.

Proof of Lemma 20

Call the vertices of c_4 $u_0, ..., u_{l(c_4)-1}$ in the positive order they appear in c_4.

Take an arbitrary vertex $u_i \in c_4$. The neighbours of u_i form a 6-cycle (call its vertices $w_{i,0}, ..., w_{i,5}$ in the positive order they appear in the cycle) because they are connected by the sides opposite to u_i of the triangles having u_i as a vertex. Since not only u_i, but also its neighbours are regular vertices, they all have exactly 3 edges remaining and for any $w_{i,j}$, these remaining vertices are forming an interval in $n_1(w_{i,j})$. And from the triangulatedness of G', for any j, the rightmost of these three neighbours of $w_{i,j}$ is the same as the leftmost one of $w_{i,j-1}$ (counted $\mod 6$), while the central one is connected with the other two. So if we name the common neighbour of $w_{i,j-1}$ and $w_{i,j}$ as $t_{i,j-1,j}$ and the common neighbour of $w_{i,j}$, $t_{i,j-1,j}$ and $t_{i,j,j+1}$ as $t_{i,j}$, then all the drawn edges exist. So although the $N_2(u_i)$ might not be isomorphic with the graph in Figure 4 because of the $t_{i,j}$'s and $t_{i,j-1,j}$'s not being regular (it is possible, that some of the vertices listed above coincide or there exist edges not shown in the drawing), a (spanning) subgraph of it can be obtained as the result of a Ψ_i function preserving incidence between vertices and edges and is also preserving triangles and orientation. Also, once we have decided the rotation of Ψ_i regarding the neighbours of $\Psi^{-1}(u_i)$, the function is unique.

Proof of Lemma 21

For all i $(i = 0, ..., l(c_4) - 1)$ let the Isbell colouring we coloured $\Psi_i^{-1}(N_2(u_i))$ with be named χ_i.

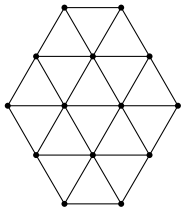


Figure 23

Similarly as in Lemma 10, there is a unique function preserving incidences between vertices and edges that projects the graph in Figure 23 into $N_2(u_i) \cap N_2(u_{i+1})$ counted $\mod l(c_4)$ for $i = 0, ..., l(c_4) - 1$. Thus, we can suppose Ψ_i and Ψ_{i+1} are the same on $N_2(u_i) \cap N_2(u_{i+1})$. And since this graph contains a subgraph that is isomorphic with that in Figure 5, from Lemma 11 we know that $\chi_i = \chi_{i+1}$. Thus, $\Psi_i^{-1}(N_2(u_i))$ is coloured with the same Isbell colouring for all $i = 0, ..., l(c_4) - 1$. And since Ψ_i is an isomorphism on $N_1(u_i)$, the $N_1(u_i)$ are also all coloured with the same Isbell colouring.

Now colour T with this colouring (call it χ_0) and define a function g from ordered pairs of colours to directions in T: for any ordered pair of colours, take the ordered pairs of neighbouring vertices of T coloured with these two colours. It is trivial from the definition of an Isbell colouring that the direction of the vector connecting the two members of such an ordered pair is uniquely defined by the ordered pair of the colours. Let this direction be the g of this pair. Also define a function h from the ordered pairs of vertices in c_4, which is defined as the g of the ordered pair of colours belonging to the particular vertices. And from $N_1(u_1)$ being coloured with χ_0, we know that $\angle h(u_{i-1}, u_i), h(u_i, u_{i+1}) = (2 - (e(u_i, int(c_4)))) \cdot \frac{\pi}{3}$, where $int(c_4)$ is defined as the side it goes around in a positive direction). And from this and $\sum_{i=0}^{l(c_4)-1} \angle(f(u_i, u_{i+1}), f(u_{i+1}, u_{i+2})) = 0$, we get that the curvature of c_4 is divisible by 6.

2048 Without Merging

Hugo A. Akitaya* Erik D. Demaine† Jason S. Ku‡
Jayson Lynch§ Mike Paterson¶ Csaba D. Tóth‖

Abstract

Imagine $t \leq mn$ unit-square tiles in an $m \times n$ rectangular box that you can tilt to cause all tiles to slide maximally in one of the four orthogonal directions. Given two tiles of interest, is there a tilt sequence that brings them to adjacent squares? We give a linear-time algorithm for this problem, motivated by 2048 endgames. We also bound the number of reachable configurations, and design instances where all t tiles permute according to a cyclic permutation every four tilts.

1 Introduction

2048 is a popular open-source video game by then-19-year-old Gabriele Cirulli [Cir14, Wik20] that took the world by storm in 2014. It was inspired by another game called 2048 by Saming, which in turn was inspired by a similar game called 1024!, which in turn was inspired by the genesis game Threes! by Vollmer, Wohlwend, and Hinson released just two months earlier, all of which inspired many other variants. See [LU18] for more on the history and descriptions of several game-variant rules. Cirulli's 2048, Threes!, Fives, Det2048, and Fibonacci all feature the same kind of movement, also identical to the 2011 physical puzzle game Tilt [BDF+19, BLC+19, BGC+20]: unit-square tiles in a rectangular box with only four global ***tilt*** controls — sliding all tiles maximally in an orthogonal direction among $\{N, E, S, W\}$. Each tile has a label, and certain labeled tiles merge together (into a single newly labeled tile) when slid into each other; the goal is generally to produce a tile with a particular label. Each game also has a (possibly randomized) algorithm for introducing new tile(s) after each move. Most of these games (in their perfect-information form) are NP-hard [LU18, AAD16].

Our results. In this paper, we consider a variant where no tile merging can happen, no additional tiles are introduced (as in [AAD16]), and there are no fixed obstacles (as in most games above, but unlike Tilt and e.g. 1024!). This minimal variant, which we call ***2048 without merging***, intends to capture some core mathematical structure of the many game variants listed above.

In particular, we study the problem of whether two particular tiles can be made adjacent by a sequence of tilt operations, which is motivated by a subproblem arising near the end of a 2048 game, where the board has two 1024-labeled tiles and the player wants to make them adjacent so that another tilt merges them into a 2048-labeled tile. This problem was posed by Mike Paterson in 2018. We solve this problem in Section 3 by giving an $O(t)$-time algorithm to decide, given an initial $m \times n$ board configuration of $t \leq mn$ tiles and two marked tiles t_1 and t_2, whether there exists a tilt sequence that brings t_1 and t_2 to adjacent squares. In the positive case, our algorithm also outputs the minimum tilt sequence. This algorithm generalizes to decide in $O(st)$ time whether any pair among s special tiles (1024s) can be made adjacent. In particular, this running time is $O(t)$ for $s = O(1)$ and always $O(t^2)$.

We also consider the combinatorial structure of the motion of all tiles, which is roughly described by powers of a single permutation. In Section 4, we give a lower bound of $2^{\Omega(\sqrt{t})}$ and an upper bound of $2^{O(\sqrt{t \log t})}$ on the number of different states that can be reached by a tilt sequence from an initial $m \times n$ board configuration with $t = \Theta(mn)$ tiles. Section 5 shows that there exist initial $m \times n$ board configurations with permutation cycles of length $\Omega(mn)$ and, for even m and n, there is a configuration in which every tile is part of the same permutation cycle. In the latter configurations, each tile can reach any possible target square via a tilt sequence of length $O(mn)$.

2 Definitions and Basics

We base our terminology on [BLC+19, BGC+20]. A ***board*** is a rectangular region of the 2D square lattice, whose 1×1 cells we refer to as ***squares***. We represent an $m \times n$ board B by $\{(x, y) \mid x \in \{0, 1, \ldots, m-1\}, y \in \{0, 1, \ldots, n-1\}\}$ where $(0, 0)$ represents the bottom-left square. Let $\mathcal{T}$ be a set of t objects called ***(slidable)***

*School of Computer Science, Carleton University, hugoakitaya@gmail.com
†CSAIL, MIT, edemaine@mit.edu
‡EECS, MIT, jasonku@mit.edu
§CSAIL, MIT, jaysonl@mit.edu
¶Dept of Computer Science, University of Warwick, UK, M.S.Paterson@warwick.ac.uk
‖CSU Northridge and Tufts University, cdtoth@eecs.tufts.edu

tiles. A ***configuration*** is an injective function $C : \mathcal{T} \to B$. We call a square ***full*** if it is in the image of C, and ***empty*** otherwise.

Tilt is an operation that takes a configuration C and a direction $d \in \{N, E, S, W\}$ and returns a configuration C' as follows. A tilt is ***horizontal*** if $d \in \{E, W\}$, or ***vertical*** if $d \in \{N, S\}$. We describe a tilt for $d = N$; the other cases are symmetric. For all rows j from top to bottom, and for all columns i, if (i, j) is full, then move the tile at (i, j) (marking (i, j) empty) to the topmost square (i, j') in the ith column marked as empty, where $j' \geq j$ (marking (i, j') as full).

A ***tilt sequence*** is a sequence of tilts applied to a configuration represented by a sequence of directions $D = (d_1, d_2, \ldots, d_k)$. Two tilt sequences are equivalent if they produce the same configuration. A row (column) is called ***monotone non-increasing*** if every full square is to the left of (below) every empty square in the row (column). ***Monotone non-decreasing*** is defined analogously. We call a configuration ***SW-canonical*** if every row and every column is monotone non-increasing. Alternatively, C is SW-canonical if a tilt with direction S or W would return C. Symmetrically, we can define NE-, NW-, and SE-canonical configurations.

Lemma 1 *After one horizontal and one vertical tilt, not necessarily in this order, we get a canonical configuration.*

Proof. Without loss of generality, we apply the tilt sequence (S, W). After the first tilt, every column is monotone non-increasing. Consequently, the number of full squares in a row is monotone (non-increasing) from bottom to top. Similarly, the second tilt makes every row monotone non-increasing. By definition, a horizontal tilt does not change the number of full squares in each row. The columns in the resulting configurations are also monotone non-increasing. The result is a SW-canonical configuration. □

The following lemma will allow us to focus only on a clockwise or counterclockwise tilt sequence, i.e., a substring of $(N, E, S, W)^*$ or $(N, W, S, E)^*$ (where the Kleene Star notation A^* denotes sequence A repeated zero or more times).

Lemma 2 *Every shortest tilt sequence between two configurations is either a clockwise or counterclockwise tilt sequence.*

Proof. It is clear by definition that the tilt sequence (S, S) is equivalent to (S). Similarly, (N, S) is equivalent to S. Then the shortest tilt sequence between configurations either has length less than 2 or one of its two first tilts is horizontal and the other one is vertical. Without loss of generality, let (S, W) be the prefix of such sequence. By Lemma 1, after these two initial tilts we get a SW-canonical configuration and hence the third tilt cannot be in the S or W direction. It cannot be E, since (S, E) is equivalent and shorter than (S, W, E). Hence the sequence starts with (S, W, N). We can then induct on the length of the sequence to show that all subsequent tilts must follow a clockwise order. □

The following lemma allows us to describe the movement of the tiles using permutations.

Lemma 3 *After every four tilts in a shortest sequence, the union of the filled squares form the same shape, but with permuted tile positions.*

Proof. Suppose C is a SW-canonical configuration where the length of row i is a_i and the length of column j is b_j. Since every row and column in C is monotone non-increasing, $a_0 \geq \ldots \geq a_{m-1}$, and we find that $b_j = |\{i \mid a_i \geq j\}|$. After a horizontal tilt in direction E to reach configuration C', all the row lengths remain as before and are monotone non-increasing. Since C' is a SE-canonical configuration the column lengths are non-decreasing and $b_{n-1-j} = |\{i \mid a_i \geq j\}|$, as before but counting from the right. So the sequence of column lengths is now reversed, i.e., $b_{m-1}, \ldots, b_0$. From C' a vertical tilt in direction N would reverse the row length order, and so on, and a complete cycle of four tilts will return row and column lengths to their original sequences. □

Let g be the permutation referred by Lemma 3. Our techniques will be based on the cyclic subgroup generated by g and its cycle decomposition. For example, if the initial configuration is given by a lower-triangular matrix in a square board where exactly the squares on and below the main diagonal are full, the permutation g will induce cycles of length 3.

If $\mathcal{C}$ is a cycle in the permutation generated by g, and $s_1, s_2 \in \mathcal{C}$, then the ***cycle index*** $\text{ind}(s_1, s_2)$ is the smallest nonnegative integer i such that $g^i(s_1) = s_2$, that is, i successive application of g carries s_1 to s_2. We say that two cycles are ***adjacent*** if there exists a pair of adjacent squares with one square in each cycle.

3 Algorithm

In this section we give an algorithm that decides the following problem. Given an initial configuration C_0 on an $m \times n$ board, and two tiles $t_1, t_2 \in \mathcal{T}$, can a tilt sequence produce a configuration C_k in which t_1 and t_2 are in adjacent squares? If yes, output a shortest such sequence. We denote by C_i the configuration obtained after the ith tilt.

1. Guess the first two tilt directions from $\{(S, W), (S, E), (N, W), (N, E), (E, N), (E, S), (W, N), (W, S)\}$, checking whether t_1 and t_2 are adjacent

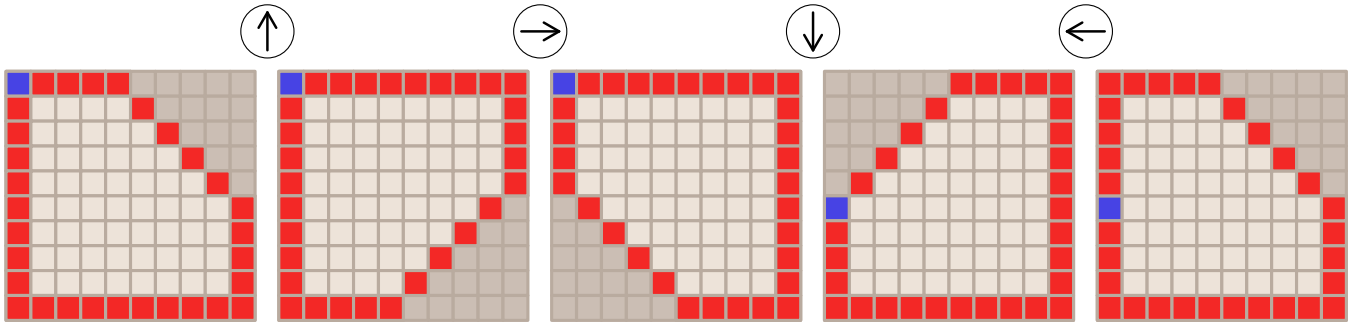


Figure 1: Application of tile sequence (N, E, S, W) on a single ring of C_n resulting in permutation $g(C_n)$. Each tile moves cyclically counterclockwise by k positions, where $2k$ is the width of the ring, e.g., the tile marked in blue.

in C_0 and C_1. The obtained configuration C_2 is in canonical form.

2. Guess a type of canonical configuration from $\{SE, SW, NE, NW\}$, computing the first configuration in the sequence, i.e., a configuration C' in $\{C_2, C_3, C_4, C_5\}$.

3. Compute the permutation g equivalent to four tilts in clockwise or counterclockwise direction depending on the guessed first two tilts.

4. Compute the cycle decomposition of g.

5. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the cycles containing t_1 and t_2 respectively. By restricting our attention to the elements of $\mathcal{C}_1$ and $\mathcal{C}_2$, successively applying g results in $\operatorname{lcm}(|\mathcal{C}_1|, |\mathcal{C}_2|)$ possible different permutations of these elements. For example, if $\mathcal{C}_1 = \mathcal{C}_2$, then the number of such states is $|\mathcal{C}_1|$. Check in each of these states whether t_1 is adjacent to t_2.

Theorem 4 *Given an initial configuration C_0 on an $m \times n$ board, and two tiles $t_1, t_2 \in \mathcal{T}$, a shortest tilt sequence required to make t_1 and t_2 adjacent can be computed in $O(|\mathcal{T}|)$ time.*

Proof. The correctness of our algorithm follows directly from Lemmas 1–3. After the first two configurations, a clockwise or counterclockwise tilt sequence produces only canonical configurations. In Step 1, the algorithm tries all possible starting tilts and both rotation directions. Step 2 considers all possible types of canonical configurations. Steps 3–5 considers all possible relative positions of t_1 and t_2 given a rotation direction of the tilt sequence and a type of canonical configuration. Hence, the algorithm is correct.

Most of the algorithm runs in $O(|\mathcal{T}|)$ time. Steps 1–2 add a $8 \cdot 4 = 32$ multiplicative factor to the runtime. Steps 3–4 can be executed in $O(|\mathcal{T}|)$ time by simulating four tilts to obtain a directed graph representing g, and obtaining the cycles of g containing t_1 and t_2 via a DFS from such vertices. Here we use that a tilt operation can be simulated in $O(|\mathcal{T}|)$ time using counting sort on the coordinates; for instance, a horizontal tilt determines the x order of tiles in each row, then moves the tiles to one extreme of the board without changing such order.

In Step 5, we need more care. A naïve implementation iterates through all $\operatorname{lcm}(|\mathcal{C}_1|, |\mathcal{C}_2|) = O(|\mathcal{T}|^2)$ different possible positions of t_1 and t_2 since both $|\mathcal{C}_1|$ and $|\mathcal{C}_2|$ are $O(|\mathcal{T}|)$. The resulting running time is $O(|\mathcal{T}|^2)$.

We can reduce the runtime of Step 5 to $O(|\mathcal{T}|)$ as follows. For all $|\mathcal{C}_1| \leq |\mathcal{T}|$ possible positions of t_1, we can try all possible adjacent squares (up to four), checking whether a tilt sequence can bring tiles t_1 and t_2 simultaneously to those squares, as follows. Suppose the desired positions are indeed in $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, say the pth and qth positions of $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, counting from the initial positions of t_1 and t_2 with zero indexing. By Bézout's Identity [JJ98], the set of all integer linear combinations $i|\mathcal{C}_1| + j|\mathcal{C}_2|$ is exactly the set of integer multiples of $d = \gcd(|\mathcal{C}_1|, |\mathcal{C}_2|)$. Thus the desired tiles can meet at the specified position exactly if $p \equiv q \pmod{d}$. We can then find the actual number of tilts by using the Chinese Remainder Theorem. Since we only need to compute d once, spending $O(|\mathcal{T}|)$ time, and we can test each of the $4|\mathcal{C}_1|$ meeting positions in constant time, the total runtime is $O(|\mathcal{T}|)$. □

Generalization to $|\mathcal{S}|$ special tiles. We can generalize Theorem 4 to the case of a subset $\mathcal{S} \subseteq \mathcal{T}$ of special tiles, where $2 \leq |\mathcal{S}| \leq |\mathcal{T}|$. Specifically, we consider the following problem. Given an initial configuration C_0 on an $m \times n$ board, and a subset $\mathcal{S} \subseteq \mathcal{T}$ of $|\mathcal{S}|$ special tiles, can a tilt sequence produce a configuration C_k in which two special tiles are in adjacent squares? If yes, output a shortest such sequence. We modify our previous algorithm, designed for the case $|\mathcal{S}| = 2$, by replacing Step 5 with the following:

5*. For each pair $\{s_1, s_2\}$ of adjacent squares that belong respectively to ***special cycles*** $\mathcal{C}_i$ and $\mathcal{C}_j$ of g (that is, cycles containing at least one special tile in $\mathcal{S}$), check whether any of the special tiles in $\mathcal{C}_i$

and $\mathcal{C}_j$ can simultaneously move to s_1 and s_2 by successively applying g, as follows:

- Let $d = \gcd(|\mathcal{C}_i|, |\mathcal{C}_j|)$.
- For each $t \in \mathcal{C}_i \cap \mathcal{S}$, compute the cycle index $\text{ind}(t, s_1)$ (as defined in Section 2), and let $I_1 = \{\text{ind}(t, s_1) \bmod d \mid t \in \mathcal{C}_i \cap \mathcal{S}\}$.
- Similarly, compute $I_2 = \{\text{ind}(t, s_2) \bmod d \mid t \in \mathcal{C}_j \cap \mathcal{S}\}$.
- Check whether $I_1 \cap I_2 \neq \emptyset$.

Theorem 5 *Given an initial configuration of $|\mathcal{T}|$ tiles on an $m \times n$ board and a set $\mathcal{S} \subset \mathcal{T}$ of special tiles, a shortest tilt sequence required to make two special tiles adjacent can be computed in $O(|\mathcal{S}| \cdot |\mathcal{T}|)$ time.*

Proof. For the correctness of the algorithm, assume that there exist two special tiles $t_1 \in \mathcal{C}_i \cap \mathcal{S}$, $t_2 \in \mathcal{C}_j \cap \mathcal{S}$ and an integer $h \geq 0$ such that $g^h(t_1) = s_1$ and $g^h(t_2) = s_2$, where s_1 and s_2 are adjacent squares in a canonical configuration C'. Then $h \equiv \text{ind}(t_1, s_1) \pmod{|\mathcal{C}_i|}$ and $h \equiv \text{ind}(t_2, s_2) \pmod{|\mathcal{C}_j|}$. By Bézout's Identity, $\text{ind}(t_1, s_1) \equiv \text{ind}(t_2, s_2) \pmod{d}$, where $d = \gcd(|\mathcal{C}_i|, |\mathcal{C}_j|)$. Conversely, if $\text{ind}(t_1, s_1) \equiv \text{ind}(t_2, s_2) \pmod{d}$, then there exits an integer $h \geq 0$ such that $h \equiv \text{ind}(t_1, s_1) \pmod{|\mathcal{C}_i|}$ and $h \equiv \text{ind}(t_2, s_2) \pmod{|\mathcal{C}_j|}$.

As noted above, Steps 1–4 of the algorithm run in $O(|\mathcal{T}|)$ time. In Step 5*, we can memoize the gcd of the lengths of all pairs of adjacent special cycles. For each such pair $\{\mathcal{C}_i, \mathcal{C}_j\}$, the Euclidean algorithm computes $\gcd(|\mathcal{C}_i|, |\mathcal{C}_j|)$ in $O(\log(|\mathcal{C}_i| + |\mathcal{C}_j|)) = O(\log |\mathcal{T}|)$ time. There are at most $|\mathcal{S}|$ special cycles, and hence $O(|\mathcal{S}|^2)$ pairs of special cycles. Furthermore, every pair of adjacent cycles contain a pair of adjacent squares, and since each square has at most four neighbors, there are $O(|\mathcal{T}|)$ adjacent pairs of squares. Therefore, the gcds can be computed in $O(\min\{|\mathcal{S}|^2, |\mathcal{T}|\} \log |\mathcal{T}|)$ time. This time bound is always $O(|\mathcal{S}| \cdot |\mathcal{T}|)$: if $|\mathcal{S}|^2 \leq |\mathcal{T}|$, then we have $O(|\mathcal{S}|^2 \log |\mathcal{T}|) = O(|\mathcal{S}| \cdot |\mathcal{S}| \log |\mathcal{T}|) = O(|\mathcal{S}| \sqrt{|\mathcal{T}|} \log |\mathcal{T}|) = O(|\mathcal{S}| \cdot |\mathcal{T}|)$; and if $|\mathcal{T}| \leq |\mathcal{S}|^2$, then we have $O(|\mathcal{T}| \log |\mathcal{T}|) = O(|\mathcal{T}| \sqrt{|\mathcal{T}|}) = O(|\mathcal{T}| \cdot |\mathcal{S}|)$.

Step 5* iterates through all $O(|\mathcal{T}|)$ pairs (s_1, s_2) of adjacent squares. Suppose that $s_1 \in \mathcal{C}_i$ and $s_2 \in \mathcal{C}_j$ where $\mathcal{C}_i$ and $\mathcal{C}_j$ are special cycles. Given the precomputed indices of g, the index sets I_1 and I_2 can be computed in $O(|\mathcal{C}_i \cap \mathcal{S}| + |\mathcal{C}_j \cap \mathcal{S}|) = O(|\mathcal{S}|)$ time. Checking whether $I_1 \cap I_2 = \emptyset$ via hashing takes $O(|I_1| + |I_2|) = O(|\mathcal{S}|)$ time. For each index in $I_1 \cap I_2$, we can then find the actual number of tilts using the Chinese Remainder Theorem, in overall $|I_1 \cap I_2| = O(|\mathcal{S}|)$ time. After memoizing the gcd of adjacent special cycles, Step 5* of the algorithm thus runs in $O(|\mathcal{S}| \cdot |\mathcal{T}|)$ time. □

4 Bounds on Reachable Configurations

Lower bound. For even n, consider the configuration

$$C_n = \begin{bmatrix} F & L \\ F & F \end{bmatrix}$$

on an $n \times n$ board where F is a full $n/2 \times n/2$ square, and L is a $n/2 \times n/2$ matrix where exactly the squares below the main diagonal are full (see Figure 1). Let the outer ***ring*** be the set of extremal tiles in N, E, S, or W direction, and define inner rings recursively. The outer ring contains $7(n/2-1)+3$ tiles, while each successive inner ring contains 7 fewer tiles. So C_n comprises $n/2$ rings, with $7k+3$ tiles in ring k for $k \in \{0, \ldots, n/2-1\}$, where the innermost ring 0 is an L-shaped tromino.

Lemma 6 *Each ring in C_n is self-contained (does not mix with adjacent rings), and the cyclic order of elements around the ring does not change after applying g.*

Proof. By symmetry, it suffices to show that each ring in C'_n, the NW-canonical configuration obtained after a N tilt from C_n, is exactly a ring in C_n, and that the cyclic order of tiles does not change. The outermost ring is composed of the first and last columns, and the extremal tiles in each remaining column. After the tilt, all such tiles remain extremal and their order along the convex hull remains the same. No other tiles become extremal because the number of tiles in adjacent columns differ by at most one. Hence, the outermost ring remains the same. If we look at the remaining tiles after removing the outermost ring, they form a configuration C_{n-2} in the $(n-2) \times (n-2)$ board obtained after removing the extremal rows and columns. The configuration C'_{n-2} obtained after a N tilt from C_{n-2} can be obtained in the same way from C'_n. Hence, the second outermost ring also remains the same in C_n and C'_n. By induction, all rings remain the same. □

Theorem 7 *The number of different configurations reachable from C_n is $e^{\Theta(n)} = e^{\Theta(\sqrt{t})}$ where t is the number of tiles in C_n.*

Proof. We first show that, after applying g, the elements in ring k (with width $2k$) shift by k positions counterclockwise along its ring. By Lemma 6 it suffices to show that the top-left tile x moves down by k. As shown in Figure 1, x does not move after the N and E tilts, moves down by k after the S tilt, and again does not move after the W tilt.

We prove the claimed bound by focusing on a subset of rings and bounding the number of different states of the tiles in the selected rings that are obtained by successively applying g. We can restrict to rings with prime lengths of the form $7k+3$. In such cases, the ring induces a single cycle in g because the length of the ring and the

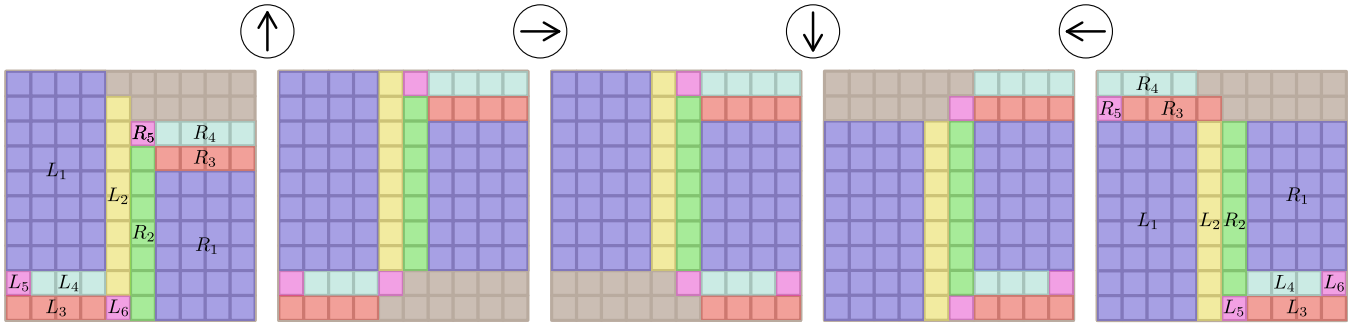


Figure 2: Application of the tilt sequence (N, E, S, W) to C_{mn} resulting in the permutation $g(C_{mn})$, depicting the movement of tile subsets $\{L_1, L_2, L_3, L_4, L_5, L_6, R_1, R_2, R_3, R_4, R_5\}$. Each of these subset moves as a rigid block.

amount that each element shifts around the ring after g are coprimes and each element will eventually visit every position in the ring. Using a variant of the prime number theorem for arithmetic progressions [Dab89, Tao09],

$$\sum_{\substack{p \le x \\ p \equiv a \bmod q}} \log p \equiv \frac{x}{\varphi(q)}(1 + o(1))$$

if $(a, q) = 1$, where φ is Euler's totient function. In particular, the product of primes $p \le n/2$, $p \equiv 3 \pmod 7$, is $e^{\Theta(n)} = e^{\Theta(\sqrt{t})}$. □

Upper bound. Starting from a SW-canonical configuration containing t tiles in an $m \times n$ board, the number of reachable configurations equals the least common multiple of the cycle lengths. Every cycle has length at most t (the number of tiles), hence the lcm of the cycle lengths is bounded above by $\mathrm{lcm}(1, 2, \ldots, t) = e^{t(1+o(1))}$.

We can improve upon this bound by realizing that the sum of the cycle lengths must be t. The maximal least common multiple of a partition of t into positive integers is known as Landau's function [Nic13], and it is asymptotically $e^{\Theta(\sqrt{t \log t})}$.

5 Long Cycles

In this section we provide a construction for an initial $m \times n$ board configuration C_{mn} with $\Theta(mn)$ tiles with a single permutation cycle. We can assume that m and n are even, or else we construct the instance of size $(2\lfloor m/2 \rfloor) \times (2\lfloor n/2 \rfloor)$ and add an empty row and/or column. Refer to Figure 2. We leave empty the last $n/2+1$ $(n/2)$ squares in the top (second to top) row of C_{mn}. The remaining squares are filled with tiles.

We now show that the permutation g induced by the tilt sequence (N, E, S, W) has a single cycle. We first describe g by giving a successor function based on the coordinates of a given square, then we describe an algorithm that outputs in order all elements in a cycle in g and show that the number of outputs equals the number of tiles. Let $s(x, y)$ be such a successor function where $s(x, y)$ is the coordinates of the square after the square (x, y) in g. Let $\Delta(x, y) = s(x, y) - (x, y)$ be the vector from (x, y) to its successor. Refer to Figure 2. We partition the occupied squares in the board into 11 regions as follows, where $p(x, y)$ denotes the region containing square (x, y):

$$p(x,y) = \begin{cases} L_1 & \text{if } 0 \le x < m/2 - 1 \text{ and } 2 \le y < n \\ L_2 & \text{if } x = m/2 - 1 \text{ and } 1 \le y < n - 1 \\ L_3 & \text{if } 0 \le x < m/2 - 1 \text{ and } y = 0 \\ L_4 & \text{if } 1 \le x < m/2 - 1 \text{ and } y = 1 \\ L_5 & \text{if } x = 0 \text{ and } y = 1 \\ L_6 & \text{if } x = m/2 - 1 \text{ and } y = 0 \\ R_1 & \text{if } m/2 + 1 \le x < m \text{ and } 0 \le y < n - 4 \\ R_2 & \text{if } x = m/2 \text{ and } 0 \le y < n - 3 \\ R_3 & \text{if } m/2 + 1 \le x < m \text{ and } y = n - 4 \\ R_4 & \text{if } m/2 + 1 \le x < m \text{ and } y = n - 3 \\ R_5 & \text{if } x = m/2 \text{ and } y = n - 3 \end{cases}$$

Now we define $\Delta(x, y)$ based on the above partition, which can be easily verified by following the tiles initially in each region after four clockwise tilts as shown in Figure 2.

$$\Delta(x,y) = \begin{cases} (0, -2) & \text{if } p(x,y) = L_1 \\ (0, -1) & \text{if } p(x,y) = L_2 \\ ((m/2 + 1), 0) & \text{if } p(x,y) = L_3 \\ (m/2, 0) & \text{if } p(x,y) = L_4 \\ (m/2, -1) & \text{if } p(x,y) = L_5 \\ (m/2, 1) & \text{if } p(x,y) = L_6 \\ (0, 2) & \text{if } p(x,y) = R_1 \\ (0, 1) & \text{if } p(x,y) = R_2 \\ (-m/2, 2) & \text{if } p(x,y) = R_3 \\ (-(m/2 + 1), 2) & \text{if } p(x,y) = R_4 \\ (-m/2, 1) & \text{if } p(x,y) = R_5 \end{cases}$$

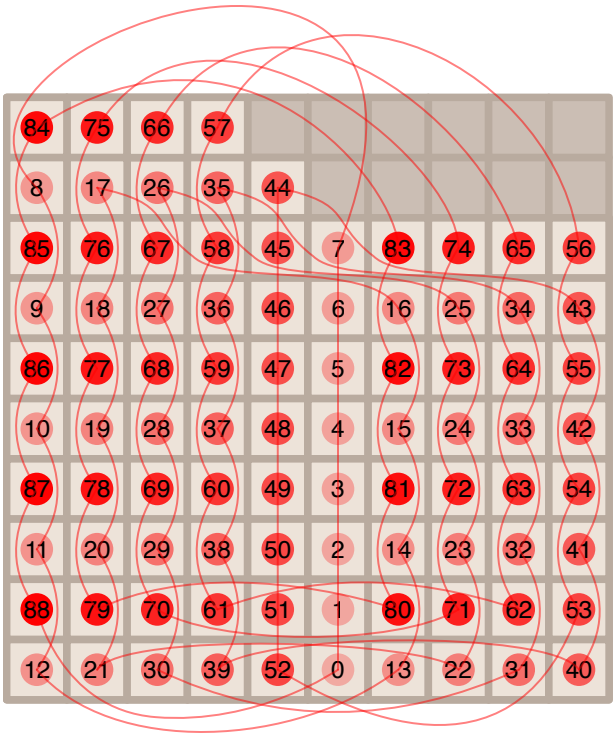


Figure 3: An example output cycle of g applied to C_{mn} for $m = n = 10$. Indexing starts with 0 at $(\frac{m}{2}, 0)$ increasing to 88 at $(0, 1)$, where increasing indices are outlined with increasing opacity.

We now describe an algorithm that outputs one cycle of g starting with square $(m/2, 0)$. Refer to Figure 3 for an example of output of the algorithm.

1. For $0 \le j < n-2$: output $(\frac{m}{2}, j)$
2. For $0 \le i < \frac{m}{2} - 1$:
   (a) For $0 \le j < \frac{n}{2}$: output $(i, n-2-2j)$
   (b) For $0 \le j < \frac{n}{2} - 1$: output $(\frac{m}{2} + 1 + i, 2j)$
3. For $0 \le j < n-1$: output $(\frac{m}{2} - 1, n-2-j)$
4. For $0 \le i < \frac{m}{2} - 1$:
   (a) For $0 \le j < \frac{n}{2} - 1$: output $(m-1-i, 1+2j)$
   (b) For $0 \le j < \frac{n}{2}$: output $(\frac{m}{2}-2-i, n-1-2j)$

Theorem 8 *The permutation g of the tilt sequence (N, E, S, W) on configuration C_{mn} has a single cycle of length $mn - m - 1$.*

Proof. It suffices to show that the algorithm above correctly outputs a cycle in order, and that it outputs all $mn - m - 1$ full squares of C_{mn}. We now focus on the former. Step 1 outputs all squares in R_2 and R_5 from bottom to top. This matches the successor function for R_2. The successor of the square in R_5 is $(0, n-2)$ which is the first position output by Step 2. Step 2(a) outputs squares in L_1 and a square in L_3 at the end of the loop, according to the successor function in L_1, i.e., two units below the previous one. The first square output by each execution of Step 2(b) is the successor of the square in L_3 output by Step 2(a), i.e., $m/2+1$ units to the right. Step 2(b) outputs squares in R_1 and a square in R_3 at the end of the loop, according to the successor function in R_1, i.e., two units above the previous one. The next output square is by either another execution of Step 2(a), or by Step 3, both obey the successor function of R_3, $\Delta(x, y) = (-m/2, 2)$. Step 3 outputs squares in L_2 from top to bottom and then outputs the square in L_6, obeying the successor function in L_2. The next square is $(m-1, 1)$ in R_1 satisfying the successor function in L_6. Step 4(a) outputs squares in R_1 and a square in R_4 at the end of the loop, according to the successor function in R_1, i.e., two units above the previous one. The first square output by each execution of Step 4(b) is the successor of the square in R_4 output by Step 4(a), i.e., $\Delta(x, y) = (-m/2-1, 2)$. Step 4(b) outputs squares in L_1 and a square in L_4 at the end of the loop or, in the last execution of the loop, it outputs the square in L_5. The order is the same as specified in L_1.

The number of outputs of the algorithm is:

$$\begin{aligned}(n-2) + (m/2-1)(n/2+n/2-1) + \\ (n-1) + (m/2-1)(n/2-1+n/2)\end{aligned} = mn-m-1,$$

which is equal to t, as desired. □

6 Open Problems

A few interesting problems in this space remain open:

1. Close the gap between $2^{\Omega(\sqrt{t})}$ and $2^{O(\sqrt{t \log t})}$ for the number of reachable configurations in 2048 without merging.

2. Are there examples where all t tiles permute in a single cycle, for even t? (Our construction works only for odd t.)

3. Can Theorem 5 be improved, that is, is it possible to decide in $o(st)$ time whether any pair of s special tiles among t total tiles in a given configuration can be made adjacent?

4. What happens in higher dimensions, such as 3D, where Lemma 2 no longer holds? (This question was posed by Martin Demaine in 2018.)

Acknowledgments

We thank Fae Charlton, Martin Demaine, and Leonie Ryvkin for helpful discussions on this topic.

References

[AAD16] Ahmed Abdelkader, Aditya Acharya, and Philip Dasler. 2048 without new tiles is still hard. In Erik D. Demaine and Fabrizio Grandoni, editors, *Proceedings of the 8th International Conference on Fun with Algorithms*, volume 49 of *LIPIcs*, pages 1:1–1:14, 2016.

[BDF+19] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Jarrett Lonsford, and Rose Morris-Wright. Particle computation: complexity, algorithms, and logic. *Natural Computing*, 18(1):181–201, 2019.

[BGC+20] Jose Balanza-Martinez, Timothy Gomez, David Caballero, Austin Luchsinger, Angel A. Cantu, Rene Reyes, Mauricio Flores, Robert T. Schweller, and Tim Wylie. Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms*, pages 2625–2641, 2020.

[BLC+19] Jose Balanza-Martinez, Austin Luchsinger, David Caballero, Rene Reyes, Angel A. Cantu, Robert T. Schweller, Luis Angel Garcia, and Tim Wylie. Full tilt: Universal constructors for general shapes with uniform external forces. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms*, pages 2689–2708, 2019.

[Cir14] Gabriele Cirulli. 2048. Github repository, 2014. https://github.com/gabrielecirulli/2048. Playable version at https://play2048.co/.

[Dab89] Hédi Daboussi. On the prime number theorem for arithmetic progressions. *Journal of Number Theory*, 31(3):243–254, 1989.

[JJ98] Gareth A. Jones and J. Mary Jones. Divisibility. In *Elementary Number Theory*, pages 1–17. Springer, London, 1998.

[LU18] Stefan Langerman and Yushi Uno. Threes!, fives, 1024!, and 2048 are hard. *Theoretical Computer Science*, 748:17–27, 2018.

[Nic13] Jean-Louis Nicolas. On Landau's function $g(n)$. In Ronald L. Graham, Jaroslav Nešetřil, and Steve Butler, editors, *The Mathematics of Paul Erdős, I*, Algorithms and Combinatorics, pages 207–220. Springer, 2013.

[Tao09] Terence Tao. The prime number theorem in arithmetic progressions, and dueling conspiracies. Blog post, 2009. https://terrytao.wordpress.com/2009/09/24/the-prime-number-theorem-in-arithmetic-progressions-and-dueling-conspiracies/.

[Wik20] Wikipedia. 2048 (video game). https://en.wikipedia.org/wiki/2048_(video_game), 2020.

External Exploration of a Convex Polygon

Kyle Clarkson* William Evans†

Abstract

In this work we consider competitive strategies for the *convex polygon inspection problem* where a mobile agent must explore the exterior of a convex polygon in the plane without prior knowledge of the polygon's shape. The agent starts at a point exterior to the polygon and can see infinitely far in all directions; however, the polygon occludes its sight. Using only what it observes, the agent must traverse a route that does not intersect the polygon to see all edges. We present two strategies for the agent to employ, the better of which guarantees that the resulting route is at most $4.5\pi \approx 14.14$ times the length of the shortest route from the starting point that sees all edges. This result is an improvement on recent work by Tiktinsky et al. [24] in which the authors derived a strategy which has length at most 89.83 times the shortest possible route. The key ideas behind our strategies make use of techniques by Icking et al. [17, 18] to optimally look around a single corner, which we extend to multiple corners of the polygon, as well as the *angle hull* of a polygon by Hoffmann et al. [15].

1 Introduction

Algorithms that consider the traversal of an entity in a region have been of interest to many researchers over the years—from the theoretical perspective in which an agent with some task is modelled as a mobile point in the plane with perfect controls and sensor readings [21, 9, 11, 7], to the application perspective in which mobile robots with uncertainty in their actions and sensors must perform various operations in the world such as infrastructure inspection [2, 3]. For such problems one is typically interested in the length of the route taken, in particular what is the shortest route the entity can take to complete the task.

We take the former perspective and consider the following problem. A mobile agent in the plane, initially positioned at a point s external to a convex polygon, can *see* any point q on the boundary of the polygon from position p if the line segment $\overline{pq}$ does not intersect any obstacle. Here, the only obstacle is the polygon itself. The goal of the agent is to see all edges of the polygon by traversing the shortest path possible in the plane; once the agent has seen all edges its task is finished. In this work we consider when the polygon's edges are unknown to the agent in advance (yet the agent knows the polygon is convex) and only revealed by moving to see them. We are interested in a strategy for the agent to see all edges such that for any polygon and starting position, the distance traversed by employing the strategy is at most a constant times the optimal (shortest) path to see all edges from the same starting position.

*Department of Computer Science, University of British Columbia, clarkson@cs.ubc.ca

†Department of Computer Science, University of British Columbia, will@cs.ubc.ca

2 Related Work

The notion of a point-agent that must see all edges of a body in the plane is a well studied problem in the area of computational geometry. The *Watchman Route* problem is concerned with such an agent who, given some starting position in the interior of a simple polygon, must traverse the interior of the polygon to see all edges and then return to the starting position, forming a cycle. Nearly four decades ago Chin and Ntafos [5] first considered this problem as a generalization of the Art Gallery Guarding problem [22]. Many others have studied this problem and its variants. Such works include considering when the polygon has holes in its interior, in which determining whether there exists a watchman route with length at most some amount is NP-complete [10] and the design of efficient algorithms for determining the shortest watchman route for simple polygons without holes [5, 6, 23]. The difference between our problem and the watchman route problem is that we do not require the route to return to its starting position and that the agent is inspecting the exterior of a single convex polygon.

An *offline algorithm* is an algorithm that has access to its entire input when it begins, such as the position of edges which compose the polygon. In contrast, an *online algorithm* is one that obtains its input during its execution. In our problem, only when the agent sees an edge will it learn the location of that edge. For this reason one does not expect an online algorithm to produce the optimal inspection path that can be found by an offline algorithm. However one can hope to construct an online algorithm which will generate an inspection path that is not too much longer than the optimal inspection path. We say that an online algorithm (a strategy) is *c-competitive* if the length of the path produced by the online algorithm is at most c times the length of the

optimal path produced by an offline algorithm for all instances of the problem and some constant (*competitive ratio*) c. In this work we are interested in finding a c-competitive strategy for the convex polygon inspection problem.

The Watchman Route problem has also been considered in the online scenario, that is, the watchman does not have knowledge of the polygon in advance and only learns it when edges are seen. We recommend the survey by Ghosh and Klein [13] which considers several exploration tasks for a mobile robot that must search in the plane. Of particular relevance to our work is how an agent should move to inspect around several corners. We discuss this problem in more depth later, but for a strategy to be competitive it must consider both paths that follow the boundary of the polygon and those that move away from the boundary to look around a sequence of corners. We consider techniques first introduced by Icking, Klein, and Ma for looking around a single corner when the angle of the corner is unknown [18]. When the angle of the corner is known the shortest path to look around the corner is a straight line segment. However when the angle is unknown the best competitive strategy follows a curved trajectory. The authors derive the optimal online strategy by solving a differential equation in polar form (a closed-form solution seems difficult to obtain). The shape of the resulting path is a curve that "spirals" towards the corner with length at most ≈ 1.212 times that of the optimal path. Dorrigiv, López-Ortiz, and Tawfik expanded on the one-corner case by considering the optimal average-case strategy where the angle of the corner is drawn from a homogeneous probability distribution [8].

For problems which involve more than one angle, approaches (including ours) use strategies based on circular arcs, which have useful geometric properties. Hoffmann et al. [16] use such an approach as part of their solution to the online watchman route problem. Here the authors give a strategy for exploring the interior of a simple, non-convex polygon which results in a cycle with length at most 26.5 times of the length of the optimal watchman route for the polygon. They base their strategy on a fundamental construction called the *angle hull* of a polygon [15].

The recent work of Tikinsky et al. achieve a competitive strategy for the convex polygon inspection problem which yields a path with length at most 89.83 times that of the optimal. This strategy works by defining the notion of scope. As the agent moves to explore the polygon, it keeps track of the unexplored region. Initially this region is unbounded, but as the agent explores it becomes bounded. This region is described by a triangle ΔuWv where u, v are vertices of the polygon and W is a point exterior to the polygon. The scope is the angle $\angle uWv$. Initially, the agent performs a spiral search to bound the region, then follows a second spiral search to reduce the region even more. Once the region is small enough the agent moves towards the point W to see the remaining edges. Both this work and ours make use of a spiral search strategy in which an agent can competitively move in both the counterclockwise and clockwise directions around the polygon to inspect its edges. This problem is an instance of the *m-lane cow-path* problem for $m = 2$, which has been studied by numerous researchers under different names and itself is a member of a family of problems called *search games* [12, 1, 19, 20].

3 Our Results

In Section 4 we give a 2π-competitive strategy for the subproblem of searching around several corners (a chain of edges) when the direction of traversal is known. Then in Section 5 we show how any c-competitive strategy for this subproblem can be incorporated into a competitive strategy for the convex polygon inspection problem by using a doubling approach to search in both directions around the polygon. When we use the strategy of Section 4 for the chain subproblem, we obtain an 18π-competitive strategy for the convex polygon inspection problem.

We present our main result in Section 6 where we expand on the work of Hoffmann et al. [15] to show that an agent which follows arcs generated by the angle hull of the chain of edges is $\pi/2$-competitive for the subproblem. By using the search procedure described in Section 5, we obtain a 4.5π-competitive strategy for the convex polygon inspection problem. In Section 7, we describe several open problems.

4 The Corner-Radius Strategy

To tackle the convex polygon inspection problem, we first consider a simpler subproblem where the agent only needs to traverse in one direction, say counterclockwise, around the polygon. Imagine that the agent must see only a chain (a consecutive subsequence) of edges from a convex polygon and that the furthest clockwise edge of the chain is seen from the agent's starting position. We call this the *convex chain* problem. If the agent knows the positions of the edges, then the shortest inspection path is simply the shortest counterclockwise collision-free path to a point on the extension of the last chain edge. However if the edge positions are not known, what is a competitive strategy for the agent to see the chain?

The crux of this problem is how can an agent look around a sequence of corners competitively. Icking et al. [18] investigated competitive strategies for looking around a single corner (from a start point both on and off the chain) and described the optimal competitive

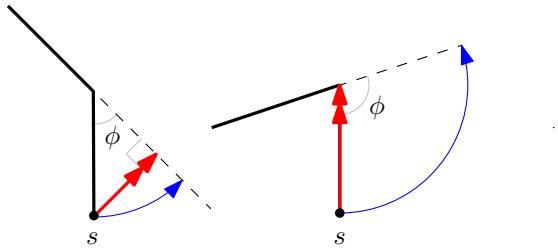


Figure 1: Two cases in which the optimal route, in red (double-headed arrow), either moves straight towards the corner or straight to the closest point on the extension of the next edge. In blue (single arrow) is a circular arc with length at most π times the length of the red arc.

strategy. Of course, a strategy to move directly to the corner may not be competitive if the angle between the next edge's extension and the current edge is acute (see Figure 1.) In such a case, the path should follow the shortest straight line segment to the next edge's extension. Since the online agent does not know this angle, it could guess that the angle is small and continuously revise its guess as it follows the optimal strategy for the assumed angle. This leads to a path that follows an arc of the circle centered at the corner, which yields a competitive ratio of at most π.

While a circular arc may work for one corner, it is yet to be shown that this approach will be competitive for several corners. We now show that the strategy of following a trajectory of circular arcs, where the circles are centered on the furthest seen vertices of the chain, results in a 2π-competitive strategy for the convex chain problem. Denote this strategy as the *corner-radius* strategy. We first assume that the starting position s is located on the chain, then consider the case where the starting position is off the chain. Without loss of generality, we further assume no edge of the chain (or polygon) is colinear with its neighbouring edges. Depending on the structure of the chain, the optimal route may follow the boundary of the chain, immediately leave the boundary from starting position s, or follow the boundary for some number of edges, then leave the boundary. Note that the optimal route leaves the boundary at a point where it can follow a perpendicular to the last edge's extension without intersecting the polygon. To show that the corner-radius strategy is competitive, we first partition the optimal path $\mathcal{O}$ into two subpaths: $\mathcal{O}_{\parallel}$ which is the part of the optimal path that follows the boundary of the chain, and $\mathcal{O}_{\perp}$ which is the part of the optimal path that is off of the boundary and meets the extension of the last edge at a right angle. Next for each edge e_i, numbered from the furthest clockwise edge, we consider how its length contributes to either $\mathcal{O}_{\parallel}$ or $\mathcal{O}_{\perp}$. Finally we show that for each edge

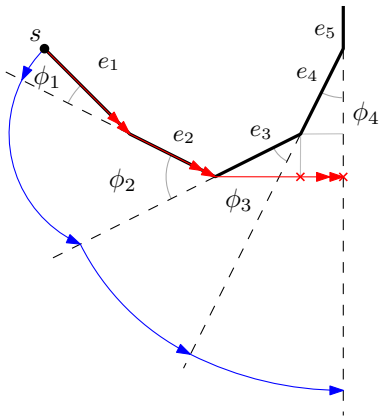


Figure 2: The corner-radius strategy for an instance of the convex chain problem. The optimal path is shown in red and the arcs of the corner-radius strategy in blue. The optimal path leaves the boundary of the chain after traversing edge e_2.

e_i, the corner-radius strategy will contribute at most 2π times the amount the optimal route contributes for edge i.

For $i = 1$ to $n-1$, let ϕ_i denote the angle between edge i and the extension of edge $i+1$. Suppose edge u, $0 \leq u < n$, is the last edge on the boundary traversed by the optimal route. (where $u = 0$ represents when the optimal path leaves the boundary at s.) By our partitioning scheme, we have

$$\begin{aligned} |\mathcal{O}| &= |\mathcal{O}_{\parallel}| + |\mathcal{O}_{\perp}| \\ &= \sum_{j=1}^{u} |e_j| + \sum_{j=u+1}^{n-1} |e_j| \sin(\phi_j + \phi_{j+1} + \cdots + \phi_{n-1}) \end{aligned} \tag{1}$$

where the terms $|e_j| \sin(\phi_j + \phi_{j+1} + \cdots + \phi_{n-1})$ come from further decomposing $\mathcal{O}_{\perp}$ into the bottom lengths of a collection of right triangles; the lengths are the projections of edges $u+1$ to $n-1$ onto the line perpendicular to the extension of the last edge with length $|\mathcal{O}_{\perp}|$; see Figure 2.

We now consider the length of the corner-radius strategy, $\mathcal{A}_{CR}$. To look around corner i, the strategy follows a circle whose center is the endpoint of edge i that is closer to the end of the chain. As s is located at the start of the chain, the radius of this circle (the distance between the agent and the corner) is the sum $|e_1| + |e_2| + \cdots + |e_i|$. Therefore we can express the length of the arc the agent traverses to look around corner i as $a_i = \phi_i(|e_i| + |e_{i-1}| + \cdots + |e_1|)$. After summing over all $n-1$ corners, we rearrange terms to isolate the contribution of each edge's length to $\mathcal{A}_{\mathcal{CR}}$ and split the

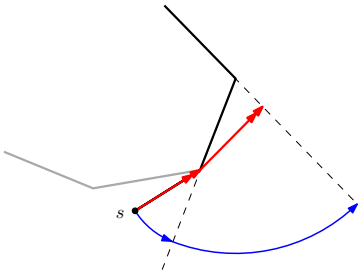


Figure 3: The starting position is off of the boundary of chain. The original chain with initially seen edges are grayed out and the new chain starting at s consists of three edges. The optimal path and the path generated by the corner-radius strategy are the same for both chains.

terms to match those in $\mathcal{O}_{\parallel}$ and $\mathcal{O}_{\perp}$:

$$|\mathcal{A}_{\mathcal{CR}}| = \sum_{i=1}^{n-1} \phi_i (\sum_{j=1}^{i} |e_j|) = \sum_{j=1}^{n-1} |e_j| (\sum_{i=j}^{n-1} \phi_i)$$
$$= \sum_{j=1}^{u} |e_j| (\sum_{i=j}^{n-1} \phi_i) + \sum_{j=u+1}^{n-1} |e_j| (\sum_{i=j}^{n-1} \phi_i)$$

We make the following observations. First since the chain originated from the boundary of a convex polygon, the sum of all angles ϕ_i is at most 2π. Second, for all edges $i > u$, $\sum_{j=i}^{n-1} \phi_j \leq \pi/2$. This follows as the optimal route leaves the boundary after traversing edge u. Third, it can easily be verified that for $x \in (0, \pi/2), x \leq \frac{\pi}{2} \sin(x) \leq 2\pi \sin(x)$. Combining the above, we have,

$$|\mathcal{A}_{\mathcal{CR}}| \leq 2\pi \sum_{j=1}^{u} |e_j| + 2\pi \sum_{j=u+1}^{n-1} |e_j| \sin(\sum_{i=j}^{n-1} \phi_i)$$
$$\leq 2\pi |\mathcal{O}|$$

Finally we consider the scenario where the starting position s is not on the boundary. Under the conditions of the convex chain problem, the agent sees the furthest clockwise edge (and perhaps other edges) of the chain from s. Suppose we form a new convex chain by replacing the edges seen from s with the segment from s to the furthest counterclockwise vertex seen from s. Observe that from s the optimal path that sees the original chain is the same optimal path that sees the new chain. Further the path generated by the strategy will be the same for both chains. By our previous analysis the corner-radius strategy achieves a competitive ratio of at most 2π for the new chain and thus the same for the original chain; see Figure 3.

We summarized the above analysis in the following statement.

Theorem 1 *The corner-radius strategy is 2π-competitive for the convex chain problem.*

5 A Two-Sided Search Strategy

We return to our original problem where the agent must inspect a polygon, yet it is not clear in what direction around the polygon the agent should traverse. From the previous section, one approach seems clear. From starting position s, use the corner-radius strategy to explore the polygon in one direction for some distance. If the agent has not finished inspecting the polygon, it returns to s retracing the arcs it traversed from s, then uses the corner-radius strategy to explore the polygon in the other direction for some larger distance than before. The agent repeats this process of searching both sides, increasing the distance it traverses each time, until the inspection is complete.

To ensure that the resulting path length is at most a constant times the optimal path's length, we use an approach for the well-known cow-path problem in which a cow, located at the origin, is searching for a gate in a fence (a point on the x-axis) located at distance $d \geq \lambda$ away. In the offline case the cow moves distance d. However in the online case the cow does not know d or if the x-coordinate of the gate is positive or negative. The question is what strategy can the cow employ such that, for all possible locations of the gate, it traverses a distance at most c times d for some constant c? Baeza-Yates at al. [1] showed that a doubling strategy where the distances traversed are $\lambda, 2\lambda, 4\lambda, \ldots$ results in a total traversal distance that is at most $9d$. The result relies on the online strategy knowing a lower bound λ on d. We describe how an agent can calculate such a lower bound given what it sees from its initial position in Section 8.2 of the Appendix. Bose et al. [4] consider how search strategies can be improved when both lower and upper bounds are specified on the distance d.

Let $\mathcal{C}$ denote a c-competitive strategy for the convex chain problem. The *two-sided, $\mathcal{C}$ strategy* (TS,$\mathcal{C}$S) for the convex polygon inspection problem uses the strategy $\mathcal{C}$ to define the path traced by the search in each direction around the polygon and doubles the distance (initially traversing distance λ) in each iteration of the search. We now show that TS,$\mathcal{C}$S is a $9c$-competitive strategy for the convex polygon inspection problem. This implies that the two-sided, corner-radius strategy is 18π-competitive. First observe that TS,$\mathcal{C}$S can be viewed as a type of cow-path problem where there are two gates, L and R, which the cow must search for, yet only realizes it has found any gate once both are discovered. The points L and R correspond to points on the extension of edges where the shortest inspection path moves in one direction around the polygon to reach L from s, then moves in the other direction from L to R ending the inspection. Second, the arcs that are generated by TS,$\mathcal{C}$S are the same arcs which are generated by two instances of the convex chain problem: one in which the agent moves in one direction from s to see edges up

to the edge whose extension L lies on, and an instance that moves in the other direction from s to see the edges up to the edge whose extension R lies on. Let P be the gate, one of L or R, that is reached last by TS,$\mathcal{C}$S, e_p denote the edge whose extension P lies on, and $|\mathcal{A}_{TS}|$ the overall distance traversed by TS,$\mathcal{C}$S. Let p denote the length of the shortest inspection path from s to see edge e_p on the corresponding convex chain instance.

The $\mathcal{C}$ strategy will generate a distance that is at most cp. As TS,$\mathcal{C}$S uses the arcs of $\mathcal{C}$ to search for P, an application of the cow-path analysis implies $|\mathcal{A}_{TS}| \leq 9cp$. Finally, the shortest inspection path that sees all edges of the polygon must reach both L and R not just whichever point P is; thus the length of the optimal path is bounded below by p, and so,

Theorem 2 *The* two-sided, *$\mathcal{C}$ strategy is $9c$-competitive for the convex polygon inspection problem, where c is the constant competitive ratio of the convex chain strategy $\mathcal{C}$.*

6 The Angle Hull Strategy

In the previous section we arrived at a competitive strategy by solving two subproblems: how to competitively search around several corners in one direction and how to search in two directions around the polygon. By employing a better strategy for the first subproblem, we can achieve a better competitive ratio for the convex polygon inspection problem. We now describe a $\pi/2$-competitive strategy for the convex chain problem which when employed in the same manner as previously described (i.e. a doubling search around both sides of the polygon) results in a 4.5π-competitive strategy for the convex polygon inspection problem.

To improve on the corner-radius strategy, we would like the agent to still circle around the chain, but to not venture too far away from it. With the corner-radius strategy the arcs generated were centered on vertices of the chain. A better approach is to place the center of such circles within the interior of the chain (that is, the side of the chain in which the agent does not traverse.) To this end our strategy uses the concept of the *angle hull*, first introduced by Hoffmann et al. [14, 15], which for a convex polygon is the set of connected, circular arcs around the polygon such that a camera with a 90 degree field of view facing the polygon would see only the boundary of the polygon with no white space; see Figure 4. Hoffmann et al. show that the length of the angle hull is at most $\pi/2$ times the length of the boundary of the polygon. The angle hull is also an inspection path for the polygon. However, since the optimal inspection path may be shorter than the length of the boundary, it is not obvious that the angle hull provides a competitive online strategy for our inspection problem. We show that it does result in a strategy that is $\pi/2$ competitive for the convex chain problem.

In the *angle hull strategy*, the agent follows a sequence of circular arcs from an initial position s such that the angle between the lines of sight from the agent to the left-most and right-most vertices of the observed chain form a right angle, until the agent sees the last edge of the chain. If the agent's initial view angle at s is not 90 degrees, we replace the edges seen from s with a single edge from s to the remaining chain as in Section 4.

Theorem 3 *The angle hull strategy is $\frac{\pi}{2}$-competitive for the convex chain problem.*

Proof. Let $e_1, e_2, \ldots, e_n$ be the sequence of edges in the convex chain with s the first vertex of the chain. The optimal path follows the chain up to the u^{th} vertex and then leaves the chain to follow a segment perpendicular to the extension of the last edge (e.g. in Figure 4, $u = 2$.) Let a_i be the i^{th} circular arc followed by the angle hull strategy with d_i the diameter of the circle connecting the extreme vertices of the subchain seen from a_i. Let $e_{i_s}, e_{i_{s+1}}, \ldots, e_{i_t}$ be the edges in this subchain. Clearly, $d_i \leq |e_{i_s}| + |e_{i_{s+1}}| + \cdots + |e_{i_t}|$. Let α_i be the angle spanned by a_i so it's length is $d_i\alpha_i$. Note: to match the notation used by Hoffmann et. al [15], α_i is measured not from the circle's center but from the circle's perimeter, which implies $0 < \alpha_i \leq \pi/2$. The i^{th} angle *belongs* to edge j if the edge can be seen from arc a_i. Let $\alpha_{j_s} + \cdots + \alpha_{j_t}$ denote the sum of angles belonging to the j^{th} edge. The length of the angle hull path is thus:

$$|\mathcal{A}_{AH}| = \sum_{i=1}^{n-1} \alpha_i d_i \leq \sum_{i=1}^{n-1} \alpha_i (|e_{i_s}| + \cdots + |e_{i_t}|)$$
$$= \sum_{j=1}^{n-1} |e_j|(\alpha_{j_s} + \cdots + \alpha_{j_t})$$

The agent will visit the extension of some chain edges twice, the first time allowing the agent to see the edge and the second when the agent can no longer see the edge (e.g. edges 1 and 2 in Figure 4.) For these edges the sum of the angles belonging to the edge is $\pi/2$.

Let u be the final edge whose extension is crossed twice by the agent. Observe that u is also the last edge in which the optimal path follows the boundary of the chain. This can be seen by noting both the optimal path and path from the strategy will terminate at the same point as both reach the extension of the last edge with a 90 degree angle field of view. For edges $j > u$, we claim that $\alpha_{j_s} + \cdots + \alpha_{j_t} = \phi_j + \cdots + \phi_{n-1}$. This follows from the fact that exactly the angles $\phi_j, \ldots, \phi_{n-1}$ belong to the edge j. In fact, $\alpha_{j_s}, \ldots, \alpha_{j_t}$ is simply a finer partition of those angles.

For example, consider edge $u + 1$. As its extension is not crossed twice, the angles that belong to edge $u + 1$

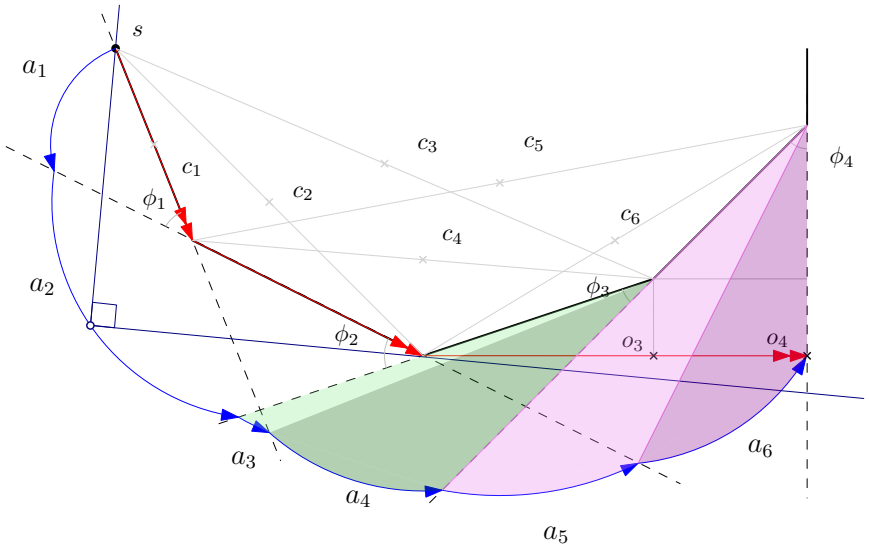


Figure 4: An instance of the angle hull strategy. The agent on arc a_2 sees both endpoints of the subchain formed by edges 1 & 2. The center of the circle of arc a_i is a cross labelled c_i. The angles α_3 and α_4 correspond to the angles in the light green and dark green shaded regions, and α_5 and α_6 for the light purple and dark purple shaded regions respectively. Note that $\alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 = \phi_3 + \phi_4$ and $\alpha_5 + \alpha_6 = \phi_4$.

are $\phi_{u+1}, \phi_{u+2}, \ldots, \phi_{n-1}$ and the angles of the arcs that can see edge $u+1$ are $\alpha_{(u+1)_s}, \ldots, \alpha_{(u+1)_t}$.

Using this relationship, we have

$$\sum_{j=u+1}^{n-1} |e_j|(\alpha_{j_s} + \cdots + \alpha_{j_t}) = \sum_{j=u+1}^{n-1} |e_j|(\phi_j + \cdots + \phi_{n-1}).$$

Thus we have

$$\begin{aligned} |\mathcal{A}_{AH}| &\leq \frac{\pi}{2} \sum_{j=1}^{u} |e_j| + \sum_{j=u+1}^{n-1} |e_j|(\phi_j + \cdots + \phi_{n-1}) \\ &\leq \frac{\pi}{2} \left(\sum_{j=1}^{u} |e_j| + \sum_{j=u+1}^{n-1} |e_j| \sin(\phi_j + \cdots + \phi_{n-1}) \right). \end{aligned}$$

Note that for $j > u$, the sum $\phi_j + \cdots + \phi_{n-1} \leq \pi/2$ and $x \leq \frac{\pi}{2}\sin(x)$ for $x \in (0, \pi/2)$. As the length of the optimal path is given by equation (1), we have $|\mathcal{A}_{AH}| \leq \frac{\pi}{2}|\mathcal{O}|$. □

Let the *two-sided, angle hull* strategy be the strategy similar to the two-sided, corner-radius strategy only that it employs the angle hull strategy instead of the corner-radius strategy. Then as described in Section 5, we have:

Theorem 4 *The two-sided, angle hull strategy is* 4.5π*-competitive for the* online polygon inspection *problem.*

7 Conclusion

We presented two competitive strategies for an agent whose task is to view the entire exterior of any unknown convex polygon from any starting position. Key to both strategies is the subproblem of how to competitively search around several corners where the agent only needs to move in one direction. We showed how the *corner-radius* strategy is 2π-competitive and the *angle hull strategy*, an extension of work done by Hoffmann et al., is $\pi/2$-competitive for this subproblem. To solve our original problem of seeing a convex polygon, we incorporated each strategy in a doubling approach, based on the two-lane cow-path problem, to explore in both directions around the polygon. We showed how our strategies were 18π-competitive and 4.5π-competitive respectively for the convex polygon inspection problem.

This is an improvement on the competitive ratio presented by Tiktinsky et al. [24], however it is unlikely to be tight. The best possible ratio for looking around one corner is ≈ 1.212 [18], which might extend to multiple convex corners and, using the doubling approach, to a ≈ 10.91-competitive strategy for the polygon inspection problem. One might also consider better ways to inspect the entire polygon than a doubling search. For example, Tiktinsky et al.'s notion of scope, which characterizes the unexplored region of the plane, might help restrict the search to improve the competitive ratio as well.

References

[1] R. A. Baeza-Yates, J. C. Culberson, and G. J. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[2] H. Baik and J. Valenzuela. Unmanned aircraft system path planning for visually inspecting electric transmission towers. *Journal of Intelligent & Robotic Systems*, 95(3-4):1097–1111, 2019.

[3] J. A. Besada, L. Bergesio, I. Campaña, D. Vaquero-Melchor, J. López-Araquistain, A. M. Bernardos, and J. R. Casar. Drone mission definition and implementation for automated infrastructure inspection using airborne sensors. *Sensors*, 18(4):1170, 2018.

[4] P. Bose, J.-L. De Carufel, and S. Durocher. Revisiting the problem of searching on a line. In *European Symposium on Algorithms*, pages 205–216. Springer, 2013.

[5] W.-P. Chin and S. Ntafos. Optimum watchman routes. In *Proceedings of the Second Annual Symposium on Computational Geometry*, pages 24–33, 1986.

[6] W.-P. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6(1):9–31, 1991.

[7] T. Danner and L. E. Kavraki. Randomized planning for short inspection paths. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 971–976, 2000.

[8] R. Dorrigiv, A. López-Ortiz, and S. Tawfik. Optimal average case strategy for looking around a corner. In *Proceedings of the 24th Canadian Conference on Computational Geometry*, pages 277–282, 2012.

[9] M. Dror, A. Efrat, A. Lubiw, and J. S. Mitchell. Touring a sequence of polygons. In *Proceedings of the 35th ACM Symposium on the Theory of Computing*, pages 473–482, 2003.

[10] A. Dumitrescu and C. D. Tóth. Watchman tours for polygons with holes. *Computational Geometry*, 45(7):326–333, 2012.

[11] S. P. Fekete, J. S. Mitchell, and C. Schmidt. Minimum covering with travel cost. *Journal of Combinatorial Optimization*, 24(1):32–51, 2012.

[12] S. Gal. Minimax solutions for linear search problems. *SIAM Journal on Applied Mathematics*, 27(1):17–30, 1974.

[13] S. K. Ghosh and R. Klein. Online algorithms for searching and exploration in the plane. *Computer Science Review*, 4(4):189–201, 2010.

[14] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. Moving an angle around a region. In *Scandinavian Workshop on Algorithm Theory*, pages 71–82. Springer, 1998.

[15] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem ii: The angle hull. Technical Report 245, Department of Computer Science, FernUniversität Hagen, 1998.

[16] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2001.

[17] C. Icking, R. Klein, and L. Ma. How to look around a corner. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, pages 443–448. University of Waterloo, 1993.

[18] C. Icking, R. Klein, and L. Ma. An optimal competitive strategy for looking around a corner. Technical Report 167, Department of Computer Science, FernUniversität Hagen, 1994.

[19] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–79, 1996.

[20] D. Kirkpatrick and S. Zilles. Competitive search in symmetric trees. In *Workshop on Algorithms and Data Structures*, pages 560–570. Springer, 2011.

[21] R. Klein. Walking an unknown street with bounded detour. *Computational Geometry*, 1(6):325–351, 1992.

[22] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

[23] X. Tan, T. Hirata, and Y. Inagaki. Corrigendum to "An Incremental Algorithm for Constructing Shortest Watchman Routes". *International Journal of Computational Geometry & Applications*, 9(03):319–323, 1999.

[24] E. Tiktinsky, S. Gul, S. Shamshanov, and R. Cohen. Online exploration outside a convex obstacle. *arXiv preprint arXiv:1807.02773*, 2018.

8 Appendix

8.1 Structure of the Optimal Path

Every polygon edge lies on a line which is called the *extension* of the edge. The edge partitions its extension into two rays: a *cw-ray* from the furthest clockwise edge's endpoint and a *ccw-ray* from the other endpoint. A ray may be crossed by an inspection path from left-to-right (facing in the direction of the ray) and/or from right-to-left.

Claim 1 *A shortest inspection path bends only at edge extensions.*

Proof. The set of edges visible from a moving point changes only when the point crosses an edge extension. Thus, by the triangle inequality, any shortest inspection path is composed of straight-line segments between crossings of edge extensions. □

Claim 2 *A shortest inspection path crosses an edge's cw-ray (or ccw-ray) in each direction at most once.*

Proof. Suppose an optimal path crosses a ray twice in the same direction. Let z be the point where the path last crosses a ray such that the path already crossed this ray in the same direction previously at some point x. Between x and z, the path must cross the ray in the opposite direction, at some point y, otherwise the path would have circumscribed the polygon between x and z and would not be optimal since it crosses the ray again at z.

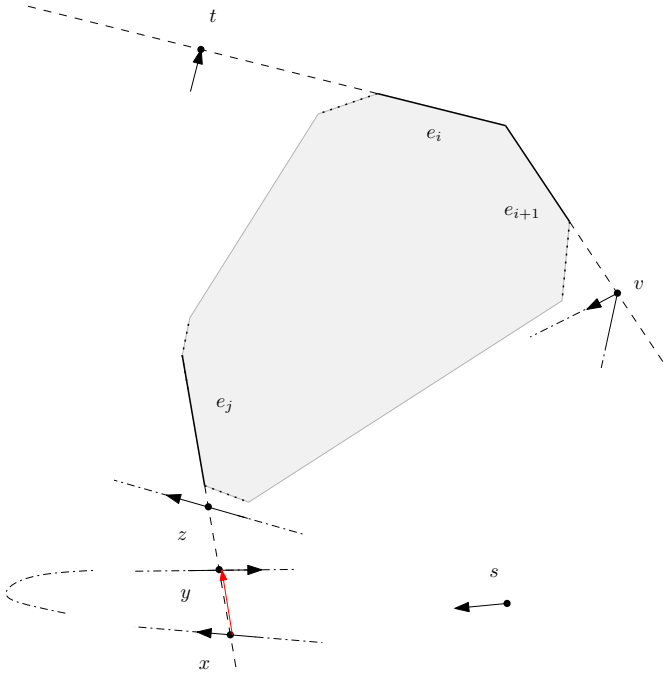


Figure 5: Edges e_i, e_{i+1}, and e_j of the polygon are shown where the ray of j is crossed twice while traversing clockwise. A shorter path can be constructed by replacing the path from x to y which does see the edge j (possibly other edges) with a path from x to y that does not see edge j, as shown in red.

Since the path continues after z, it finishes inspection by seeing the final edge i at position t while continuing to traverse in this direction. In order to see edge i, it cannot recross the ray containing z again (or else z would be chosen as this crossing). Let $i+1$ be the edge that shares a common endpoint with edge i and would be seen if the path continued to move in this direction from z. As the path ended upon seeing edge i, this implies that edge $i+1$ was seen previously in between points y and z. Let this point on the extension of $i+1$ be v. Since the polygon is convex, in order to see edge i the path from v, through z, and to t must see all the edges seen by the subpath from x to y as well - see Figure 5. This implies that the subpath from x to y is redundant and can be replaced by the segment parallel to $\overline{xy}$ which does not cross the ray of j, shortening the path. A contradiction. □

These two claims imply:

Theorem 5 *A shortest inspection path of a convex polygon P starting from a point outside P consists of a clockwise subpath around P followed by a counterclockwise subpath around P, or vice versa. Either of these subpaths may be empty.*

8.2 Initial Step Size

Let S be the subchain of edges seen from the agent's initial position at s. Let a and b be the open endpoints of S. To determine a lower bound on the length of the optimal path we return to the notion of scope defined by Tiktinsky et al. [24]. The rays $\overrightarrow{sa}$ and $\overrightarrow{sb}$ (along with the line $\overline{ab}$) define the unknown region in which the remaining edges lie. For all starting positions s, this region is unbounded, and any inspection path must close this region by first making these boundaries parallel to one another, then collapsing them such that the unknown region becomes empty (i.e. all edges are seen.)

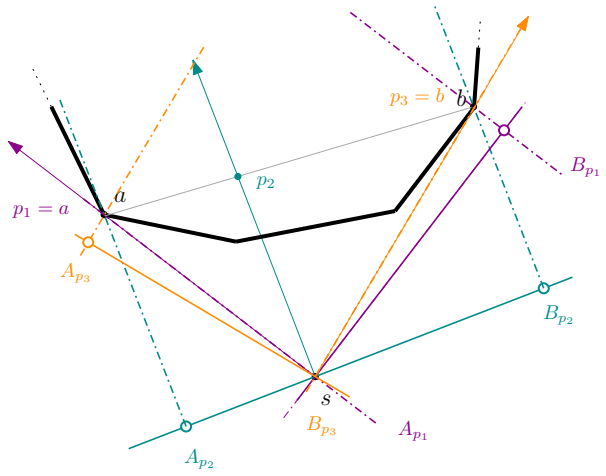


Figure 6: Three rays through $p_1 = a$, p_2, and $p_3 = b$. For each ray, the lines parallel to the ray are shown as dash dotted lines.

To determine a lower bound on the optimal we ask what is the shortest path from s that will make the boundaries of the unknown region parallel. We determine such a quantity as follows. We cast a collection of rays from s through a point $p \in \overline{ab}$, where p walks from a to b. For each such ray $\overrightarrow{sp}$, let A_p and B_p be the lines parallel to $\overrightarrow{sp}$ that pass through a and b respectively; see Figure 6. To make the boundaries parallel the agent needs to move from s to A_p and B_p. As both must be reached, the distance of traversal is at least the maximum of the two. Let $d(s, L)$ be the shortest distance from s to a line L. Thus we are interested in minimizing the quantity over points p to determine a lower bound on the optimal:

$$\mathcal{O} \geq \min_{p \in \overline{ab}} \max\{d(s, A_p), d(s, B_p)\}\}.$$

Since $\max\{d(s, A_p), d(s, B_p)\} \geq \frac{1}{2}(d(s, A_p) + d(s, B_p))$ and the latter term is at least the distance between the two lines A_p and B_p, which, for such rays, is minimized when $p = a$ or $p = b$, we have

$$\mathcal{O} \geq \min_{p \in \{a,b\}} \left(\frac{d(A_p, B_p)}{2}\right)$$

where $d(A_p, B_p)$ is the minimum distance between the two lines A_p and B_p. Note that in the last inequality we only need to cast two rays and compute the correspond lines A_a and A_b for the two values of p. Therefore the agent can efficiently compute the lower bound.

City Guarding with Limited Field of View

Ovidiu Daescu * Hemant Malik†

Abstract

Drones and other small unmanned aerial vehicles are starting to get permission to fly within city limits. While video cameras are easily available in most cities, their purpose is to guard the streets at ground level. Guarding the aerial space of a city with video cameras is a problem that so far has been largely ignored.

In this paper, we present bounds on the number of cameras needed to guard a city's aerial space (roofs, walls, and ground) using cameras with 180° range of vision (the region in front of the guard), which is common for most commercial cameras. We assume all buildings are vertical and have a rectangular base. Each camera is placed at a top corner of a building.

We considered the following two versions: (i) buildings have an axis-aligned ground base and, (ii) buildings have an arbitrary orientation. We give necessary and sufficient results for (i), necessary results for (ii), and conjecture sufficiency results for (ii). Specifically, for (i) we prove a sufficiency bound of $2k + \lfloor \frac{k}{4} \rfloor + 4$ on the number of vertex guards, while for (ii) we show that $3k + 1$ vertex guards are sometimes necessary, where k is total number of buildings in the city.

1 Introduction

Drones and other small unmanned aerial vehicles (UAVs) are already allowed to experimentally fly within city limits. For example, in August 2019, Uber announced it has selected the city of Dallas to experiment with flying drones and small UAVs, within the city. Monitoring the aerial space of big cities is thus becoming a critical problem that yet has to be addressed. Video cameras are easily available in most cities, but their purpose is to guard the streets at ground level. Guarding the aerial space of a city with cameras is a problem that has been largely ignored.

City guarding is related to the famous *art gallery problem* [29] and its many variations [41] studied in the past few decades. In almost all these studies, the art gallery lies in the plane (2D), assuming a polygonal shape with or without holes. In the art gallery problem, the goal is to determine the minimum number of point guards sufficient to see every point of the interior of a simple polygon. A point q is visible to guard g if the line segment joining q and g lies completely within the polygon. When the guards are restricted to vertices of the polygon only, they are referred to as vertex guards.

In the orthogonal art gallery problem, all edges of the polygon are either horizontal or vertical. In some versions of the art gallery problem, the polygon is allowed to have h holes. When guarding such polygons, it is allowed to place the guards at the vertices of the enclosing polygon and the vertices of the holes.

For guarding an *orthogonal polyhedron* point guards are less effective. There exist examples of polyhedra with n vertices where guards placed at every vertex do not cover the whole interior of the polyhedra; instead $O(n^{3/2})$ non-vertex guards are required [34].

The problem is also related to the following problem [8]: Given k pairwise disjoint isothetic rectangles in a plane, place vertex guards on rectangles such that every point in free space (plane area excluding the interior of the quadrilaterals) is visible to at least one guard.

The city guarding problem was introduced in [4], and is a 2.5D variant of the 2D orthogonal art gallery with holes. The input consists of k buildings, within an area bounded by an axis-parallel rectangle (this can be relaxed to the whole plane, assuming cameras have unlimited distance visibility), with each building being vertical and having an axis parallel rectangular base (a vertical rectangular prism), and the goal is to place the minimum number of guards that can see in any direction (referred as 360° field of vision), at the top corners (vertices) of some buildings, to guard the aerial space of the city. The height of a building is a strictly positive real number. In [4], they consider three variations of city guarding: (i) *Roof Guarding:* determine the minimum number of vertex guards required to guard the roofs (ii) *Ground and Wall Guarding:* determine the minimum number of vertex guards necessary to guard the ground and the walls, and (iii) *City Guarding:* determine the minimum number of vertex guards required to guard the (aerial space of the) city, which means the roofs, walls, and the ground. As with the 2D art gallery problem ([2]), the 2.5D city guarding problems are NP-hard and, by a simple reduction, so are the corresponding versions studied in this paper.

We consider the three variations of the city guarding problem with a restriction on the visibility range of the guards. Specifically, a guard is only able to see the region in front of it, i.e., the range of vision of a guard is bounded by 180°, instead of the 360° in [4]. This corresponds to the capabilities of most commercial cameras.

*The University of Texas at Dallas, daescu@utdallas.edu
†The University of Texas at Dallas, malik@utdallas.edu

In all our proofs each guard is placed at the top corner of a building and is oriented such that the seen and unseen regions of the guard are separated by a vertical plane parallel with one of the sides of the building where the camera is placed. Thus, when building bases are isothetic (axis-aligned) rectangles, a camera will face in one of four directions: East, West, North, or South (E, W, N, S). From now on, we assume cameras are placed as stated here, unless otherwise specified, and may omit mentioning camera orientation throughout the paper.

The two versions we consider are: (A) Buildings have an axis-aligned rectangular base (isothetic rectangles), (B) Buildings have a (arbitrary oriented) rectangular base. To solve the two versions, we address the following variations of the art-gallery problem:
(V1) Given an axis-aligned rectangle P with k disjoint axis-aligned rectangular holes, place vertex guards on hole boundaries such that every point inside P is visible to at least one guard, where the range of vision of a guard is 180°.
(V2) Given an axis-aligned rectangle P with k disjoint (arbitrary oriented) rectangular holes, place vertex guards on hole boundaries such that every point inside P is visible to at least one guard, where the range of vision of a guard is 180°.

For the first problem (V1), we prove a sufficiency bound of $2k + \lfloor \frac{k}{4} \rfloor + 4$ on the number of vertex guards. To obtain this bound, we provide a novel, divide and conquer algorithm. For the second problem (V2), we show that $3k+1$ vertex guards are sometimes necessary and conjecture that the bound is tight.

A comparison of our sufficiency and necessity results with those in [4] is shown in Table 1. Our solutions set an essential foundation for monitoring drones flying within city limits, using video cameras.

	[4] Guard vision range: 360° (axis-aligned rectangle buildings)	Guard vision range: 180° (axis-aligned rectangle buildings)	Guard vision range: 180° (non-axis-aligned rectangle buildings)
Roof Guarding	$\lfloor \frac{2(k-1)}{3} \rfloor + 1$	k	k
Ground and Wall Guarding	$k + \lfloor \frac{k}{4} \rfloor + 1$	$2k + \lfloor \frac{k}{4} \rfloor + 4$	$3k + 1$
City Guarding	$k + \lfloor \frac{k}{2} \rfloor + 1$	$2k + \lfloor \frac{k}{4} \rfloor + 4$	$3k + 1$

Table 1: Sufficient and necessary results comparisons. A tight bound is shown in blue color, a sufficiency bound in red color and a necessary bound in green color.

Related Work. Guarding polygons, with or without holes, has a long history in computational geometry. For guarding an orthogonal polygon with n vertices, Kahn et al. [28] showed that $\lfloor \frac{n}{4} \rfloor$ vertex guards are occasionally necessary and always sufficient. O'Rourke [32] showed that $1 + \lfloor \frac{r}{2} \rfloor$ vertex guards are necessary and sufficient to guard the interior of an orthogonal polygon with r reflex vertices. Later, Castro and Urrutia [18] provided a tight bound of $\lfloor \frac{3(n-1)}{8} \rfloor$ on the number of orthogonal guards placed on the vertices, sufficient to cover an orthogonal polygon with n vertices.

O'Rourke [33] proved that any orthogonal polygon with n vertices and h (orthogonal) holes can always be guarded with $\lfloor \frac{n+2h}{4} \rfloor$ vertex guards. Hoffmann [24] proved that $\lfloor \frac{n}{3} \rfloor$ vertex guards are always sufficient to guard an orthogonal polygon with n vertices and arbitrary number of holes. Later in 1998, Abello et al. [1] provided a tight bound of $\lfloor \frac{3n+4(h-1))}{8} \rfloor$ for the number of guards placed at the vertices of an orthogonal polygon with n vertices and h holes.

In 1994, Blanco et al. [8] considered the problem of guarding the region of the plane excluding the interior of n quadrilateral holes (obstacles). Given n pairwise disjoint quadrilaterals in the plane whose convex hull has no cut-off quadrilaterals, they showed that $2n$ vertex guards are always sufficient to cover the free space and all guard locations can be found on $O(n^2)$ time. If the quadrilaterals are isothetic rectangles, the guards can be placed in $O(n)$ time. These results directly apply to problems (V1) and (V2) we address in this paper, and they immediately imply that $4n$ guards with 180° vision always suffice, and those guards can be found in $O(n)$ and $O(n^2)$ time, respectively.

In 2008, Bao et al. [4] proposed the city guarding problem where one is given a city with k vertical buildings, each having an axis-aligned rectangular base. The guards are to be placed only at the top vertices of the buildings. They showed that $\lfloor \frac{2(k-1)}{3} \rfloor + 1$ vertex guards are sometimes necessary and always sufficient to guard the roofs (Roof Guarding Problem). They further proved that $k + \lfloor \frac{k}{4} \rfloor + 1$ vertex guards are always sufficient to guard the ground and the walls, and $k + \lfloor \frac{k}{2} \rfloor + 1$ vertex guards are always sufficient to guard the aerial space, which includes all roofs and walls of the buildings, and the ground. Their results directly apply to problem (V1) and imply that $2k + 2\lfloor \frac{k}{4} \rfloor + 2$ vertex guards with 180° vision are always sufficient to guard the walls and ground, and $2k + 2\lfloor \frac{k}{2} \rfloor + 2$ are needed to guard the city.

It follows from Table 1 that our results are a significant improvement over those that can be inferred from [8, 4]. Due to space constraints, we refer the reader to Appendix A for detailed related work.

We start with the following theorem, that allows us to limit our attention to guarding the roofs and walls of the buildings, and the ground.

Theorem 1 *If guards are placed so that the roofs, walls, and the ground of the city are guarded, then every point in the aerial space of the city is guarded.*

Proof. Let p be a point in the aerial space of the city and assume p is not guarded. Let p' be the vertical projection of p onto the ground (or a building roof) and let g be a guard that sees p' (such g exists since the ground of the city is guarded). Then g, p and p'

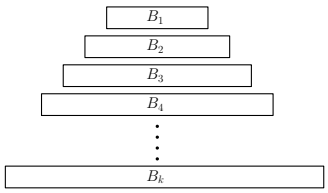


Figure 1: City Setup

define a vertical plane π. Consider the vertical triangle defined by g, p, and p'. If any portion of a building intersects the triangle side gp at some point q then the line segment qq' is part of that building, where q' is the vertical projection of q onto the ground. Since the line segment qq' intersects the triangle side gp' it then follows that p' is not visible from g, a contradiction. □

A similar result holds if we aim for walls and ground guarding only (no roof guarding requirement), including the space between the buildings.

2 Axis-Aligned Rectangle-Base Buildings

Given a rectangular city with k disjoint vertical buildings, each having an axis-aligned rectangular base, the goal is to place the minimum number of cameras that can see only the half-space in front of them (denoted as 180° range of vision), at the top corners (vertices) of the buildings to guard the city (roofs, walls, ground, and aerial space). Thus, when a guard (camera) is aligned with a wall of the building it is placed on the half-space seen by the guard is bounded by a vertical plane containing that wall.

2.1 Roof Guarding

Theorem 2 *Given a city with k disjoint axis-aligned rectangular buildings, k vertex guards are always sufficient and sometimes necessary to guard the roofs.*

Proof. The sufficiency bound is trivial (place one guard on each roof). For the necessary part, consider a set $S = \{B_1, B_2, B_3, \ldots, B_k\}$ of k buildings, as shown in Figure 1, with the following setup: (i) the height h_{B_i} of building B_i is greater than the height h_{B_j} of building B_j, $\forall\, i, j$ such that $1 \leq i < j \leq k$, and (ii) $\forall\, i < j-1$, building B_{j-1} totally blocks the visibility between B_i and B_j. In such situation, we need one guard to cover the roof of each building (details in Appendix B). □

2.2 Ground and Wall Guarding

Vertically projecting the city on the ground results in a rectangle polygon with k rectangular holes. As mentioned earlier, the number of guards required to guard the walls and ground is no larger than the number of guards needed for the following problem:

SubProblem 1 *(V1) Given an axis-aligned rectangle P with k disjoint axis-aligned rectangular holes, place vertex guards on hole boundaries such that every point inside P is visible to at least one guard, where the range of vision of guards is* 180°.

On the other hand, it is easy to see that a lower bound on the number of guards for Subproblem 1 can be used to obtain a lower bound for guarding the walls and the ground of a city with k rectangular buildings: map the holes to buildings of the same height.

It is worth noticing though that the two problems are not equivalent, that is, for a given input, fewer guards might be needed to guard the walls and ground than the number needed to guard the holes defined by projecting the buildings to the ground.

Observe that $2k + \lfloor \frac{k}{2} \rfloor + 2$ vertex guards, placed on holes, can be obtained from [4] by replacing a 360° guard with two 180° guards. In what follows, we show how to improve this bound. For each hole, extend the right vertical edge in the upward (North) direction through the interior of the polygon until it encounters some horizontal edge of a hole or the outer rectangle. After extending the vertical edges, extend both horizontal edges of each hole in the left (West) direction through the interior of the polygon until it encounters some vertical edge or extended vertical edge of a hole or the outer rectangle. The steps above divide the polygon into $2k+1$ shapes. Each shape corresponds to a monotone staircase (both in x and y-direction). Only one guard is required to guard each staircase, placed at the South-East corner of the staircase, facing West. Out of $2k+1$ guards, $2k$ guards are placed on the vertices of the holes while one guard is placed on a vertex of the rectangle P. Refer to Figure 2 for visual details.

Theorem 3 $2k+1$ *guards are always sufficient to guard the walls and ground of a rectangular city with k disjoint axis-aligned rectangular buildings, with at most one guard placed at a corner of the bounding rectangle.*

If however, we do not allow a guard to be placed at a corner of the enclosing rectangle P, then the number of guards needed could increase significantly. In the rest of this section, we prove an upper bound on the number of vertex guards, placed only on vertices of the holes (the setup in [8, 4]).

Let S be the set of k buildings in the city, contained in the axis-aligned rectangle P defined by the points

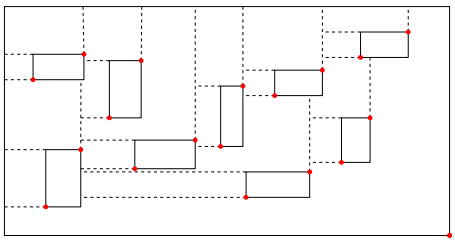
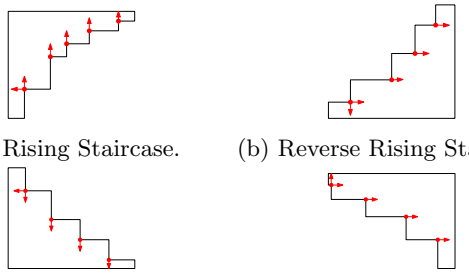


Figure 2: Extension of the right vertical edge and horizontal edges of each hole. $2k+1$ guards are always sufficient to guard the walls and ground.



(a) Rising Staircase. (b) Reverse Rising Staircase.

(c) Reverse Falling Staircase (d) Falling Staircase

Figure 3: Type of staircases and placement of guards.

$[0, 0; x, y]$. Let x_s^M, x_f^M be the starting and finishing boundry sequence along x-axis and y_s^M, y_f^M be the starting and finishing boundry sequence along y-axis. We define four types of staircases (see Figure 3): (i) Rising staircase (RS): $x_s^M = 0$, $y_f^M = y$, x_f^M is non decreasing along the positive y-axis, and y_s^M is non decreasing along the positive x-axis (ii) Falling staircase (FS): $x_f^M = x$, $y_f^M = y$, x_s^M is non increasing along the positive y-axis, and y_s^M is non increasing along the positive x-axis (iii) Reverse rising staircase (RRS): $x_f^M = x$, $y_s^M = 0$, x_s^M is non decreasing along the positive y-axis, and y_f^M is non decreasing along the positive x-axis (iv) Reverse falling staircase (RFS): $x_s^M = 0$, $y_s^M = 0$, x_f^M is non increasing along the positive y-axis, and y_f^M is non increasing along the positive x-axis

A rising staircase is constructed as follows: extend the horizontal edges of each hole towards the right (East) direction, then extend the vertical edges towards the South direction. The closed orthogonal polygon formed by the top edge and the left edge of P, and the extended edges of the holes, corresponds to a rising staircase. Falling, reverse rising, and reverse falling staircases are constructed similarly. Note that for each staircase a reflex vertex corresponds to a vertex of a hole. Thus, the number of buildings involved in a staircase is equal to the number of reflex vertices on the staircase.

We find the staircase comprising the minimum number of buildings. WLOG assume the staircase involving the minimum number of buildings is RRS (otherwise, we can rotate the input so that the staircase corresponds to RRS). For this staircase, place a guard on each reflex vertex, facing right (East), and an additional guard on the first (bottom) stair, with the guard facing down (South), as shown in Figure 3. The number of guards required to cover the staircase is one more than the number of steps (stairs) in it. In the worst case, each of the four staircases must have the same number of stairs, otherwise we can use one with the smallest number as RRS.

In what follows, we provide a divide and conquer approach to find an upper bound on the number of guards. For some building B, the vertical span of B is the parallel strip defined by the vertical sides of B and containing

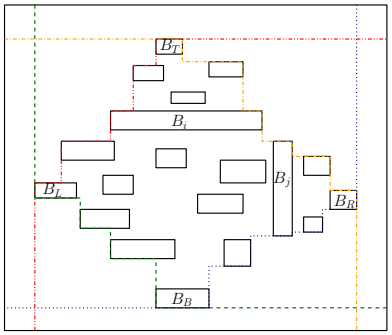


Figure 4: Staircases RS (dash dot / red), FS (dash dot / orange), RRS (dotted / blue) and RFS (dashed / green). Buildings above B_i lie in its vertical span and buildings right of B_j lie in its horizontal span.

B. The horizontal span is defined accordingly. Let B_L be the leftmost building, B_T be the topmost building, B_R be the rightmost building, and B_B be the bottommost building of the city. A building B_i is called an *internal building* if $B_i \notin \{B_L, B_T, B_R, B_B\}$. The pair of staircases (i) RS, FS (ii) FS, RRS (iii) RRS, RFS and (iv) RFS, RS are called *adjacent staircases* while the pairs (v) RS, RRS, and (vi) FS, RFS are called *opposite staircases*.

Assume two adjacent staircases, say RS and FS, share the same building $B_i \notin \{B_L, B_T, B_R, B_B\}$. Then, all buildings that lie in the upper half-plane defined by the line supporting the upper horizontal edge of B_i (buildings above B_i) are in the vertical span of B_i (see Figure 4). Similarly, if staircases RRS and FS include the walls of the same building $B_j \notin \{B_L, B_T, B_R, B_B\}$, then all buildings that lie on the right of B_j are in the horizontal span of B_j.

Notice it is possible that, from the set of building pairs $(B_L, B_T), (B_T, B_R), (B_R, B_B)$, and (B_B, B_L), the pair in one of the sets corresponds to the same building. In this situation (call it Case 0), one of the staircases consists of only one stair, and two guards are required to guard the staircase. Using a placement of guards like in Theorem 3, $2k + 2$ guards are required to cover the walls and ground of such a rectangular city. Thus, from now on, we assume this is not the case.

Consider the four staircases RS, FS, RFS, and RRS. We can have four cases:

Case 1: No adjacent pair of staircases share the same internal building, and no opposite pair of staircases share the same building.

WLOG assume that RRS contains the minimum number of stairs; place the guards in a similar fashion as in Theorem 3. Overall, we place two guards on each building and the rest on the reflex vertices of the staircase RRS.

The upper bound on the number of vertex guards required to cover the staircase of P is achieved when

the number of buildings involved in the construction of each staircase is the same. Let the staircases RS and RRS contain δ distinct buildings. Staircase FS contains $\delta - 2$ distinct buildings because building B_T is already counted in staircase RS and building B_R is counted in staircase RRS. Similarly, RFS contains $\delta - 2$ distinct buildings. Note that $\delta + \delta + \delta - 2 + \delta - 2 = k$ and thus $\delta = \lfloor \frac{k}{4} \rfloor + 1$. Therefore, to guard each staircase, we require $\delta + 1 = \lfloor \frac{k}{4} \rfloor + 1 + 1 = \lfloor \frac{k}{4} \rfloor + 2$ guards.

We place 2 guards on each building and $\lfloor \frac{k}{4} \rfloor + 2$ guards to cover the staircase. Therefore, $2k + \lfloor \frac{k}{4} \rfloor + 2$ guards are required to cover the walls and ground.

Case 2: At least one pair of opposite staircases shares the same building, and no adjacent staircases share the same interior building.

Let the staircases RS and RRS share a building B_i. Refer to Figure 5 and note that extending the top edge of B_i to the left until it hits P will not result in an intersection with the other buildings. Similarly, extending the bottom edge of B_i to the right until it hits P will not result in an intersection with the other buildings. We divide the city into two sub-cities, $city_1$ and $city_2$ (green and orange boundaries in Figure 5), by extending the top edge of B_i towards left and the bottom edge towards the right. Let B_i be included in both sub-cities.

All buildings in $city_1$ lie either above or towards the right of B_i. We place two guards on each building, one at the North-West corner and one at the South-East corner, both facing East. We further place a third guard on the North-West corner of B_i, facing West. All buildings in $city_2$ lie either below or towards the left of B_i. We place two guards on each building, one at the South-East corner and one at the North-West corner, both facing West. We further place an additional guard on the South-East corner of B_i facing East. In total, we have placed six guards on B_i. However, two guards are duplicates, so we only have four guards on B_i. Thus, $2k+2$ guards are required to cover the walls and ground of the city.

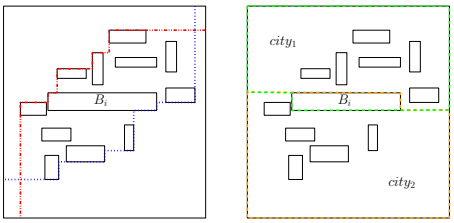
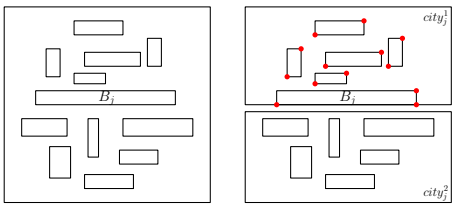


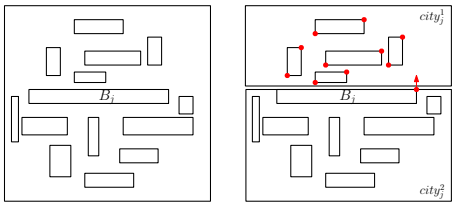
Figure 5: Building B_i is shared by staircases RS and RRS. Staircase RS is shown in dash dot / red color and staircase RRS is shown in dotted / blue color.

Case 3: At least one pair of adjacent staircases shares the same interior building, and no pair of opposite staircases shares the same building.

We use the following recursive approach to compute an upper bound on the number of guards. Let B_i be a building shared by two adjacent staircases, say RS and FS, as shown in Figure 6. Let α_i be the number of buildings that lie above B_i and β_i be the number of buildings (excluding B_i), that lie in the lower half-plane defined by the line supporting the upper horizontal edge of B_i. Note that $\alpha_i + \beta_i + 1 = k$. Let C be the set containing all such buildings B_i (walls included in more than one staircase). Let $B_j \in C$ be the building that minimizes the value $|\alpha_j - \beta_j|$, such that $\alpha_j, \beta_j \geq 3$.



(a) No buildings within the horizontal span of B_j



(b) There exist buildings within the horizontal span of B_j

Figure 6: Building B_j shares its walls with RS and FS. Divide city into two sub-cities, $city_j^1$ and $city_j^2$.

If such building does not exist then each building B_j in set C has $\alpha_j \leq 3$ or $\beta_j \leq 3$. We can use a similar argument as the one discussed in Case 2. Recall that in the worst case all staircases should have an equal number of buildings/stairs. It is easy to notice that there exist at most three buildings shared by a staircase pair (i) RS, FS (ii) FS, RRS (iii) RRS, RFS, or (iv) RFS, RS, as $\alpha_i < 3$ or $\beta_i < 3$. Let each of RS, RRS contain δ distinct buildings. Staircases FS, RFS contain $\delta - 6$ distinct buildings, as three buildings are included in each of RS and RRS. There are k buildings, $2\times\delta+2\times(\delta-6) = k$, thus $4\times\delta-12 = k$, and $\delta = \lfloor \frac{k}{4} \rfloor + 3$. To guard the staircase we need at most $\lfloor \frac{k}{4} \rfloor + 4$ guards, resulting in $2k + \lfloor \frac{k}{4} \rfloor + 4$ guards overall.

If there exists a building B_j such that $\alpha_j, \beta_j \geq 3$, we proceed as follows. Let the staircases RS and FS share building B_j. There can be two cases: (i) there are no buildings within the horizontal span of B_j (refer to Figure 6a) and (ii) there exist buildings within the horizontal span of B_j (refer to Figure 6b).

Consider the first case where none of the buildings lie within the horizontal span of B_j. We divide the city into two sub-cities, $city_j^1$ with α_j buildings and $city_j^2$ with β_j buildings. All the buildings in one of the sub-cities lie

inside the vertical span of B_j. Let this sub-city be $city_j^1$. We add B_j to $city_j^1$, which results in a total of $(\alpha_j + 1)$ buildings in $city_j^1$, and follow Case 0, which results in a total of $2(\alpha_i + 1) + 1$ guards to guard the walls and ground of this sub-city, as shown in Figure 6a. For $city_j^2$, we consider the Case it falls in and place the guards accordingly. For the placement of guards, we treat the two sub-cities as independent cities. It is important to notice that only one of $city_j^1$ and $city_j^2$ above can be in Case 3, while the other one is in Case 0. Thus, only one of the two cities could need further divisions.

Assume Case 3 keeps occurring, and we need to divide the city m times. During each division, the sub-city with corresponding building B_j contains greater than or equal to four buildings and one additional guard is required to guard such sub-city. Let the first division divide the city into two sub-cities with $k_1, k - k_1$ buildings each. The second division splits the sub-city with $k - k_1$ buildings into $k_2, k - k_1 - k_2$ buildings and so on, down to sub-cities with $k_m, k - \sum_{i=1}^{m} k_i$ buildings. Each resulting sub-city does not need to be divided further. Let $k' = \sum_{i=1}^{m} k_i$. For each sub-city with $k_1, k_2, \ldots k_m$ buildings, we require $2k_i + 1$ guards, where $k_i \geq 4$, and for the last sub city we need at most $4 + \lfloor (k - k')/4 \rfloor$ additional guards. Thus, the total number of guards required to guard the city is:
$2k_1+1+2k_2+1+\cdots+2k_m+1+2(k-k')+4+\lfloor \frac{k-k'}{4} \rfloor =$
$2k+m+4+\lfloor \frac{k-k'}{4} \rfloor \leq 2k+4+k'/4+\lfloor \frac{k-k'}{4} \rfloor \leq 2k+\lfloor \frac{k}{4} \rfloor+4$

Consider the second case where buildings lie within the horizontal span of B_j. We divide the city into two sub-cities, $city_j^1$ with α_j buildings and $city_j^2$ with β_j buildings, such that all buildings in one of the sub-cities lie inside the vertical span of B_j. Let this sub-city be $city_j^1$. We add B_j to $city_j^2$, which results in a total of $(\beta_j + 1)$ buildings in $city_j^2$. For the placement of guards, $city_j^1$ is in Case 0, and we place two guards on each building of $city_j^1$, and one guard on building B_j facing towards $city_j^1$, which results in a total of $2(\alpha_i)+1$ guards to cover the walls and ground of $city_j^1$, as shown in Figure 6b. For $city_j^2$, we consider the case it falls in and places the guards accordingly. For the placement of guards, we treat the two sub-cities as independent cities. Using a similar explanation as in the first case, we obtain the upper bound of $2k + \lfloor \frac{k}{4} \rfloor + 4$.

Case 4: At least one pair of opposite staircases and one pair of adjacent staircases share the same building.

We place guards according to Case 2 and conclude that $2k + 2$ guards are sufficient to cover the walls and ground of the city.

The derived upper bound obviously holds when the guards can have *arbitrary* orientation and we have:

Theorem 4 *$2k + \lfloor \frac{k}{4} \rfloor + 4$ guards are always sufficient to guard the walls and ground of a rectangular city with k disjoint axis-aligned rectangular buildings, with all guards placed on vertices of the buildings.*

We also obtain the following Theorem, restricting the visibility of guards in [8] to 180°:

Theorem 5 *Given k pairwise disjoint isothetic rectangles in the plane, $2k + \lfloor \frac{k}{4} \rfloor + 4$ vertex guards are always sufficient to guard the free space.*

2.3 City Guarding

Theorem 6 *Given a city with k disjoint axis-aligned buildings within an axis-aligned rectangle P, the number of axis-aligned cameras with 180° range of vision needed to guard the city (roofs, walls, and ground) is upper bounded by (i) $2k + \lfloor \frac{k}{4} \rfloor + 4$ when cameras are placed on buildings only, and (ii) $2k + 1$ when at most one camera can be placed at a corner of P.*

Proof. The solution described earlier in Section 2.2 established either $2k + \lfloor \frac{k}{4} \rfloor + 4$ guards in case (i) and or $2k + 1$ guards in case (ii) to cover the ground and the walls of the city. In both (i) and (ii), on each building, we place at least two guards, on diagonal corners, facing in the same direction. Thus, one of these two guards also guards the roof of the building. Therefore, $2k + \lfloor \frac{k}{4} \rfloor + 4$ guards are always sufficient to guard the city in case (i) and $2k + 1$ in case (ii). □

3 Arbitrary Oriented Rectangle Buildings

Given a rectangular city with k vertical buildings, each having a rectangular base, the goal is to place cameras with 180° range of vision, at the top corners (vertices) of the buildings, to guard the city. In all our proofs, each guard is aligned with a wall of the building it is placed on, similar to the axis-aligned version. The same city structure in Theorem 2 leads to:

Theorem 7 *Given a city with k disjoint rectangular buildings, k vertex guards are always sufficient and sometimes necessary to guard the roofs.*

As before, the problem of guarding the ground and the walls reduces to:

SubProblem 2 *(V2) Given an axis-aligned rectangle P with k disjoint rectangular holes, place vertex guards on hole boundaries such that every point in P is visible to at least one guard, where the range of vision of guards is* 180°.

Theorem 8 *$3k + 1$ vertex guards are sometimes necessary to guard a rectangular polygon P with k disjoint rectangular holes, where guards are placed only at vertices of the holes.*

For guarding the city we have:

Theorem 9 *$3k+1$ vertex guards are sometimes necessary to guard a city with k vertical buildings with rectangular base, where guards are placed only at the top vertices of the buildings.*

Proofs of Theorem 8 and Theorem 9 are given in Appendix C. We conjecture the bounds are tight.

References

[1] James Abello, Vladimir Estivill-Castro, Thomas Shermer, and Jorge Urrutia. Illumination of orthogonal polygons with orthogonal floodlights. *International Journal of Computational Geometry & Applications*, 8(01):25–38, 1998.

[2] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The art gallery problem is $\exists\mathbb{R}$-complete. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 65–73, 2018.

[3] Alok Aggarwal. The art gallery theorem: its variations, applications and algorithmic aspects. 1984.

[4] Lichen Bao, Sergey Bereg, Ovidiu Daescu, Simeon Ntafos, and Junqiang Zhou. On some city guarding problems. In *International Computing and Combinatorics Conference*, pages 600–610. Springer, 2008.

[5] VHF Batista. On the complexity of the edge guarding problem. In *Proc. 26th European Workshop on Computational Geometry, Dortmund, Germany, 2010*, pages 53–56, 2010.

[6] N Benbernou, Erik D Demaine, Martin L Demaine, Anastasia Kurdia, Joseph O'Rourke, Godfried Toussaint, Jorge Urrutia, and Giovanni Viglietta. Edge-guarding orthogonal polyhedra. In *Proceedings of the 23rd Canadian Conference on Computational Geometry*, pages 461–466, 2011.

[7] Iliana Bjorling-Sachs and Diane L. Souvaine. An efficient algorithm for guard placement in polygons with holes. *Discrete & Computational Geometry*, 13(1):77–109, 1995.

[8] Gregoria Blanco, Hazel Everett, Jesus Garcia Lopez, and Godfried Toussaint. Illuminating the free space between quadrilaterals with point light sources. In *PROC. COMPUTER GRAPHICS INT.,. WORLD SCIENTIFIC*. Citeseer, 1994.

[9] Édouard Bonnet and Tillmann Miltzow. An approximation algorithm for the art gallery problem. *arXiv preprint arXiv:1607.05527*, 2016.

[10] Prosenjit Bose, David Kirkpatrick, and Zaiqing Li. Worst-case-optimal algorithms for guarding planar graphs and polyhedral surfaces. *Computational Geometry*, 26(3):209–219, 2003.

[11] Javier Cano, Csaba D Tóth, and Jorge Urrutia. Edge guards for polyhedra in three-space. In *Proceedings of 24th Canadian Conference on Computational Geometry (CCCG)*, pages 163–167, 2012.

[12] Vasek Chvatal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[13] Richard Cole and Micha Sharir. Visibility problems for polyhedral terrains. *Journal of symbolic Computation*, 7(1):11–30, 1989.

[14] Jurek Czyzowicz, Eduardo Rivera-Campo, and Jorge Urrutia. Illuminating rectangles and triangles in the plane. *Journal of Combinatorial Theory, Series B*, 57(1):1–17, 1993.

[15] Jurek Czyzowicz, Eduardo Rivera-Campo, Jorge Urrutia, and Joseph Zaks. Protecting convex sets. *Graphs and Combinatorics*, 10(2-4):311–321, 1994.

[16] Jesús García López de la Calle. *Problemas algorítmico-combinatorios de visibilidad*. PhD thesis, Universidad Politécnica de Madrid, 1995.

[17] Pedro J de Rezende, Cid C de Souza, Stephan Friedrichs, Michael Hemmer, Alexander Kröller, and Davi C Tozoni. Engineering art galleries. In *Algorithm Engineering*, pages 379–417. Springer, 2016.

[18] Vladimir Estivill-Castro and Jorge Urrutia. Optimal floodlight illumination of orthogonal art galleries. In *CCCG*, pages 81–86, 1994.

[19] H Everett and G Toussaint. On illuminating isothetic rectangles in the plane,", 1990.

[20] Hazel Everett and Eduardo Rivera-Campo. Edge guarding a triangulated polyhedral terrain. In *CCCG*, pages 293–295, 1994.

[21] Steve Fisk. A short proof of chvátal's watchman theorem. *J. Combinatorial Theory (B)*, 24:374, 1978.

[22] Frank Hoffmann. On the rectilinear art gallery problem. In *International Colloquium on Automata, Languages, and Programming*, pages 717–728. Springer, 1990.

[23] Frank Hoffmann, Michael Kaufmann, and Klaus Kriegel. The art gallery theorem for polygons with holes. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 39–48. IEEE, 1991.

[24] Frank Hoffmann and Klaus Kriegel. A graph-coloring result and its consequences for polygon-guarding problems. *SIAM Journal on Discrete Mathematics*, 9(2):210–224, 1996.

[25] C Iwamoto and T Kuranobu. Improved lower and upper bounds of face guards of polyhedral terrains. *IEICE Trans. Inf. & Syst.(Japanese Edition)*, 95:1869–1872, 2012.

[26] Chuzo Iwamoto, Junichi Kishi, and Kenichi Morita. Lower bound of face guards of polyhedral terrains. *Information and Media Technologies*, 7(2):737–739, 2012.

[27] J. Urrutia J. Czyzowicz. Personal communication. 1996.

[28] Jeff Kahn, Maria Klawe, and Daniel Kleitman. Traditional galleries require fewer watchmen. *SIAM Journal on Algebraic Discrete Methods*, 4(2):194–206, 1983.

[29] Victor Klee. Is every polygonal region illuminable from some point? *The American Mathematical Monthly*, 76(2):180–180, 1969.

[30] Horst Martini and Valeriu Soltan. Survey paper. *Aequationes Math*, 57:121–152, 1999.

[31] TS Michael and Val Pinciu. Guarding orthogonal polygons with h holes: A bound independent of h.

[32] Joseph O'Rourke. An alternate proof of the rectilinear art gallery theorem. *Journal of Geometry*, 21(1):118–130, 1983.

[33] Joseph O'Rourke. Galleries need fewer mobile guards: a variation on chvátal's theorem. *Geometriae Dedicata*, 14(3):273–283, 1983.

[34] Joseph O'Rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

[35] Thomas C Shermer. Recent results in art galleries. *PROCEEDINGS-IEEE*, 80:1384–1384, 1992.

[36] Diane L Souvaine, Raoul Veroy, and Andrew Winslow. Face guards for art galleries. In *Proceedings of the XIV Spanish Meeting on Computational Geometry*, pages 39–42. Citeseer, 2011.

[37] Cs D Tóth. Art galleries with guards of uniform range of vision. *Computational Geometry*, 21(3):185–192, 2002.

[38] Csaba D Tóth. Art gallery problem with guards whose range of vision is 180. *Computational Geometry: Theory and Applications*, 3(17):121–134, 2000.

[39] Csaba D Tóth. Guarding disjoint triangles and claws in the plane. *Computational Geometry*, 25(1-2):51–65, 2003.

[40] L Fejes Tóth. Illumination of convex discs. *Acta Mathematica Hungarica*, 29(3-4):355–360, 1977.

[41] Jorge Urrutia. Art gallery and illumination problems. In *Handbook of computational geometry*, pages 973–1027. Elsevier, 2000.

[42] Giovanni Viglietta. Reprint of: Face-guarding polyhedra. *Computational Geometry: Theory and Applications*, 48(5):415–428, 2015.

[43] Giovanni Viglietta. Optimally guarding 2-reflex orthogonal polyhedra by reflex edge guards. *arXiv preprint arXiv:1708.05469*, 2017.

[44] Paweł Żyliński. Orthogonal art galleries with holes: A coloring proof of aggarwal's theorem. *the electronic journal of combinatorics*, 13(1):20, 2006.

A Related Work

A.1 Simple Polygons Results

Given a simple polygon P in the plane, with n vertices, Chvatal [12] proved that $\lfloor n/3 \rfloor$ vertex guards are always sufficient and sometimes necessary to guard P. Chvatal's proof was later simplified by Fisk [21] using the existence of a three-coloring of a triangulated polygon.

When the view of the guard is limited to 180°, Toth [38] showed that $\lfloor \frac{n}{3} \rfloor$ point guards are always sufficient to cover the interior of P (thus, moving from 360 to 180 range of vision keeps the same sufficiency number). F. Santos conjecture that $\lfloor \frac{3n-3}{5} \rfloor \pi$ vertex guards are always sufficient and occasionally necessary to cover any polygon with n vertices. Later in 2002, Toth [37] provided a lower bound on the number of point guards when the range of vision α is less than 180°. When $\alpha < 180°$, there exist a polygon P that cannot be guarded by $\frac{2n}{3} - 3$ guards. For $\alpha < 90°$ there exist P that cannot be guarded by $\frac{3n}{4} - 1$ guards, and for $\alpha < 60°$ there exist P where the number of guards needed to cover P is at least $\lfloor \frac{60}{\alpha} \rfloor \frac{(n-1)}{2}$. In 2016, Bonnet and Miltzow [9] provided an $O(log\ OPT)$ -approximation algorithm for the placement of points guards that entirely cover P.

A.2 Orthogonal Polygons Results

In 1983, Kahn et al. [28] showed that if every pair of adjacent sides of the polygon form a right angle, then $\lfloor \frac{n}{4} \rfloor$ vertex guards are occasionally necessary and always sufficient to guard a polygon with n vertices.

In 1983, O'Rourke [32] showed that $1 + \lfloor \frac{r}{2} \rfloor$ vertex guards are necessary and sufficient to cover the interior of an orthogonal polygon with r reflex vertices. Castro and Urrutia [18] provided a tight bound of $\lfloor \frac{3(n-1)}{8} \rfloor$ on

the number of orthogonal guards placed on the vertices, sufficient to cover an orthogonal polygon with n vertices.

A.3 Polygon with Holes Results

For a polygon P with n vertices and h holes, the value n is the sum of the number of vertices of P and the number of vertices of the holes. Let $g(n,h)$ be the minimum number of point guards and $g^v(n,h)$ be the minimum number of vertex guards necessary to cover any polygon with n vertices and h holes.

O'Rourke [33] gave a first proof on guarding polygons with holes and showed that $g^v(n,h) \leq \lfloor \frac{n+2h}{3} \rfloor$. Shermer conjectured that $g^v(n,h) \leq \lfloor \frac{n+h}{3} \rfloor$ and this is a tight bound. He was able to prove that, for $h = 1$, $g^v(n,1) = \lfloor \frac{n+1}{3} \rfloor$. However, for $h > 1$ the conjecture remains open. Shermer's result can be found in [34, 35].

Sachs and Souvaine [7] and Hoffmann et al. [23] showed that no art gallery problem with n vertices and h holes requires more than $\lfloor \frac{n+h}{3} \rfloor$ point guards and provided an $O(n^2)$ algorithm to find such placement, which is based on triangulation and 3-coloring.

A.4 Orthogonal Polygon with Holes Results

For this version, all polygons and holes are orthogonal and axis-aligned. Let $orth(n,h)$ be the minimum number of point guards and $orth^v(n,h)$ be the minimum number of vertex guards necessary to guard any orthogonal polygon with n vertices and h holes. Note that $orth(n,h) \leq orth^v(n,h)$.

O'Rourke's method extends to show that: $orth^v(n,h) \leq \lfloor \frac{n+2h}{4} \rfloor$. Shermer [34] conjectured that $orth^v(n,h) \leq \lfloor \frac{n+h}{4} \rfloor$ which Aggarwal [3] established for $h = 1$ and $h = 2$. Zylinski [44] showed that $\lfloor \frac{n+h}{4} \rfloor$ vertex guards are always sufficient to guard any orthogonal polygon with n vertices and h holes, provided that there exists a quadrilateralization whose dual graph is a cactus.

O'Rourke also conjectured that $orth(n,h)$ is independent of h: $orth(n,h) = \lfloor \frac{n}{4} \rfloor$, which was verified by Hoffmann [22]. In 1990, Hoffmann [22] showed that $\lfloor \frac{n}{4} \rfloor$ point guards are always sufficient and sometimes necessary to guard an orthogonal polygon with n vertices and an arbitrary number of holes. In 1996, Hoffmann and Kriegel [24] showed that $\leq \lfloor \frac{n}{3} \rfloor$ vertex guards are sufficient to watch the interior of an orthogonal polygon with holes.

Consider $orth^v(n,.)$ as the maximum of $orth^v(n,h)$ over all h. Hoffmann conjectured that $orth^v(n,.) \leq \lfloor \frac{2n}{7} \rfloor$, disproving the earlier conjecture of Aggarwal [3] that $orth^v(n,.) \leq \lfloor \frac{3n}{11} \rfloor$. In 2013, Michael and Pinciu [31] improved this bound and showed that an orthogonal gallery with n vertices and an unspecified number of holes can be guarded by at most $\frac{17n-8}{52}$ vertex guards ($17 / 52 = 0.3269$).

In 1998, Abello et al. [1] provided a first tight bound of $\lfloor \frac{3n+4(h-1))}{8} \rfloor$ for the number of orthogonal guards placed at the vertices of an orthogonal polygon with n vertices and h holes which are sufficient for the cover the polygon and described a simple linear-time algorithm to find the guard placement for an orthogonal polygon (with or without holes).

In 2016, Rezende et al. [17] showed the chronology of developments, and compared various current algorithms aiming at providing efficient implementations to obtain optimal, or near-optimal, solutions.

A.5 Families of Convex Sets (Triangles and Quadrilaterals) on the Plane Results

In 1977, Toth [40] considered the following problem: Given a set F of n disjoint compact convex sets in a plane, how many guards are sufficient to cover every point in the boundary of each set in F. Toth proved that $max\{2n, 4n-7\}$ point guards are always sufficient to cover n disjoint compact convex sets in a plane. Everett and Toussaint [19] proved that the families of n disjoint squares $n > 4$, can always be guarded with n point guards. For families of disjoint isothetic rectangles (rectangles are *isothetic* if all their sides are parallel to the coordinate axes.), Czyzowicz et al. [14] proved that $\lfloor \frac{4n+4}{3} \rfloor$ point guards suffice and conjectured that $n + c$ point guards would suffice, c is a constant. If the rectangles have equal width, then $n+1$ point guards suffice, and $n - 1$ point guards are occasionally necessary. Refer [30] for more details.

In 1994, Blanco et al. [8] considered the problem of guarding the region of the plane, excluding the interior of the quadrilaterals (free space). Given n pairwise disjoint quadrilaterals in the plane whose convex hull has no cut-off quadrilaterals, they showed that $2n$ vertex guards are always sufficient to cover the free space and all locations could be found on $O(n^2)$ time. If the quadrilaterals are isothetic rectangles, all locations can be placed in $O(n)$ time.

Urrutia [41] showed that any family of n disjoint rectangles can be guarded with at most $n+1$ point guards. A big rectangle encloses the elements of F, and consider this as an orthogonal polygon with holes. The total number of vertices is now $4n + 4$. Using the results from guarding orthogonal polygon with holes, this can be guarded with $n + 1$ guards.

Garcia-Lopez [16] proved that $\lfloor \frac{5m}{9} \rfloor$ vertex lights are always sufficient and $\lfloor \frac{m}{2} \rfloor$ vertex guards are occasionally necessary to guard the free space generated by a family of disjoint polygons with m vertices. To cover the free space generated by any family of n disjoint quadrilaterals, he proved that $2n$ vertex lights are always sufficient and occasionally necessary and that $\lfloor \frac{5n+3}{3} \rfloor$ point guards are always sufficient. He conjectured that $n + c$ point lights can always cover the free space generated

by m disjoint quadrilaterals, c is a constant which was proved false by Czyzowicz and Urrutia [27].

Czyzowicz et al. [15] proposed the following problem: Given a set F of n disjoint compact convex sets in a plane, how many guards are sufficient to protect each set in F. A set F is protected by a guard g if at least one point in the boundary of F is visible from g. They prove that $\lfloor \frac{2(n-2)}{3} \rfloor$ point guards are always sufficient and occasionally necessary to protect any family of n disjoint convex sets, $n > 2$. To protect any family of n isothetic rectangles, $\lceil \frac{n}{2} \rceil$ point guards are always sufficient, and $\lfloor \frac{n}{2} \rfloor$ point guards are sometimes necessary.

Czyzowicz et al. [14] showed that any family of n disjoint triangles can be guarded with at most $\lfloor \frac{4n+4}{3} \rfloor$ point guards are sufficient and $n-1$ are occasionally necessary to guards. They also showed that $n+1$ guards are always sufficient and $n-1$ guards are occasionally necessary to illuminate any family of n homothetic triangles and conjectured that there is a constant c such that $n+c$ point guards sufficient to guard any collection of n triangles. Later, Toth [39] showed that $\lfloor \frac{5n+2}{4} \rfloor$ guards can monitor the boundaries and the free space of n disjoint triangles.

A.6 Polyhedral Terrain Results

A polyhedral terrain is a polyhedral surface in three dimensions such that its intersection with any vertical line is either empty or a point. A polyhedral terrain is triangulated if each of its faces is a triangle. Notice that a polyhedral terrain has a different structure than a city with vertical buildings. The results related to guarding polyhedral terrain focus on edge and face guards [20, 13, 5, 10, 26, 25].

A.7 Polyhedron Results

A polyhedron in R^3 is a compact set bounded by a piece-wise linear manifold. For guarding an *orthogonal polyhedral*, points guards are less effective. There exist polyhedra with n vertices where guards placed at every vertex do not cover the whole interior and $O(n^{3/2})$ non-vertex guards are required for interior coverage [34]. Thus, results related to guarding R^3 polyhedrons focus on edge and face guards [41, 11, 36, 42, 6, 43].

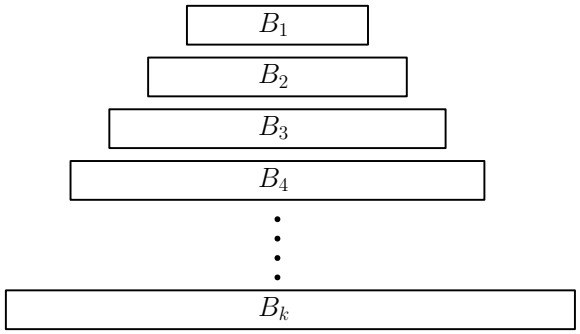
A.8 City Guarding Results

In 2008, Bao et al. [4] proposed the city guarding problem where one is given a rectangular city with k vertical buildings, each having a rectangular base. The guards are to be placed only at the top vertices of the buildings. They showed that $\lfloor \frac{2(k-1)}{3} \rfloor + 1$ vertex guards are sometimes necessary and always sufficient to guard the roofs (Roof Guarding Problem), $k + \lfloor \frac{k}{4} \rfloor + 1$ vertex guards are always sufficient to cover the ground and the walls, and $k + \lfloor \frac{k}{2} \rfloor + 1$ vertex guards are always sufficient to cover the city aerial space, which means all roofs and walls of the buildings, and the ground.

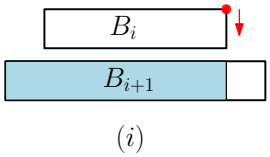
B Proof of Theorem 2

Proof. The sufficiency bound is trivial. For the necessary part, consider a set $S = \{B_1, B_2, B_3, \ldots, B_k\}$ of k buildings, shown in Figure 7a, with the following setup:

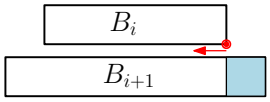
1. the height h_{B_i} of building B_i is greater than the height h_{B_j} of building B_j, $\forall\, i, j$ such that $1 \leq i < j \leq k$, and
2. $\forall\ i < j - 1$, building B_{j-1} totally blocks the visibility between B_i and B_j.



(a) City Setup



(i)



(ii)

(b) Possible guards position

Figure 7: k guards are needed to guard the roofs.

We need to place the first guard on building B_1 to guard its roof, because $h_{B_1} > h_{B_i}, \forall i > 1$. There are four possible positions to place a vertex guard on building B_1. Let the guard be placed on one of the right vertices (placing the guard on one of the left vertices results in symmetric cases). Consider the two relevant orientations of the guard, shown in Figure 7b, out of three possible positions that can see the roof of B_1, where the arrow corresponds to the direction in which the guard is guarding the roof. Notice that a guard facing West is placed at the lower vertex of B_1 rather than at the top vertex. Due to the limited visibility of the guard, there

is no vertex on building B_1 from where roofs of both building B_1 and B_2 are completely visible. Therefore, the next guard should either be placed on building B_2 or on B_1, such that the roof of building B_2 is completely visible after placing this guard. If we place the second guard on B_1, then the next guard must be placed on building B_3 because no point on the roof of building B_3 is visible by the previously placed guards. Thus, we can place the next guard on building B_2. The rest follows by induction on the number of buildings, as we are left with a similar problem on $k-1$ buildings. □

C Necessity Proof of Theorem 8

Necessity: For the necessity part, consider the input in Figure 8, with the following properties:

1. B_i lies within the span of B_j $\ ,\forall j < i$.
2. None of the edges of B_i is partially or completely visible from any vertex of B_j, $\forall\ j < i-1$ and from any vertex of B_m, $\forall\ m > i+1$.
3. From each potential position of a vertex guard on B_i, the guard is able to see at most one edge of B_{i+1}.
4. There is no guard position on B_i from where an edge of B_{i-1} and an edge of B_i are visible.

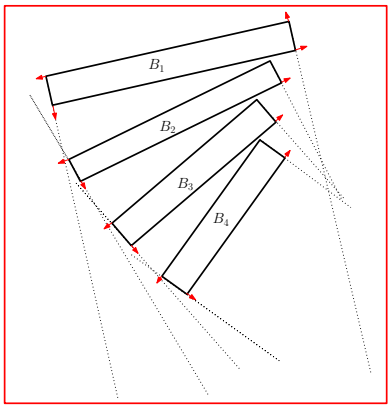


Figure 8: City Structure where $3k+1$ guards are necessary to guard the polygon

Consider the space $\wp_i$ between two consecutive holes B_i and B_{i+1}, as shown in Figure 9. Because of property 2, $\wp_i$ is not visible to any guard placed on building B_j where $j \in [1,i) \cup (i+1,k]$. Therefore, $\wp_i$ is only visible to the guards either placed on B_i or B_{i+1}. It is easy to notice that there are twelve possible guard positions on B_i and B_{i+1} from where $\wp_i$ is visible (partially from each position, see Figure 9), and these guards cover three walls, one wall of B_i and two walls of B_{i+1}. Note that these guards do not cover any other wall (partially or entirely), and the mentioned three walls are not visible (partially or entirely) to any other potential guard. Out of twelve possible guard positions, the minimum number of guards required to guard $\wp_i$ is two (either both placed on B_i or one placed on B_i and another on B_{i+1}. Therefore the space between any two consecutive holes is only guarded by the guards placed on these holes, and the minimum number of guards required to guard such space is two.

Consider the left wall, w_L^i of hole B_i. Because of the structure of the city, w_L^i is not visible to any guard placed on B_j for $j \neq i$. Hence, w_L^i can only be guarded by a guard placed on B_i, and there are four possible guard positions from where w_L^i is visible(see Figure 9). However, none of these guards positions cover any other wall in the city. Therefore, we need one guard to cover the left wall of each hole.

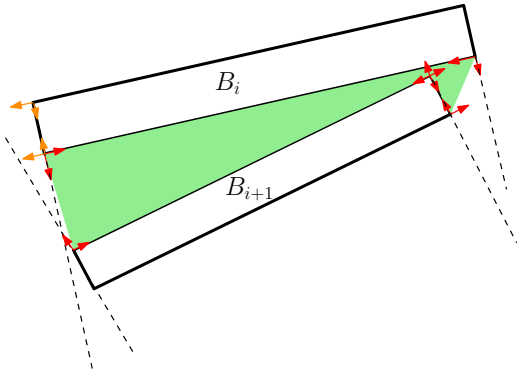


Figure 9: $\wp$ is shaded in green. Potential guard positions to cover $\wp$ are shown in red and potential guard positions to cover the w_L^i is shown in orange.

There are $k-1$ spaces $\wp_i$ in total, between consecutive buildings ($i = 1, 2, \ldots, k-1$). Thus, $2(k-1)$ guards are needed to guard their union. As argued, each left edge of a hole needs an additional guard, resulting in $3k-2$ guards. The top and right edges of B_1 and the bottom edge of B_k are not guarded, so in total, at least $3k+1$ guards are needed.

A $3k+1$ guard placement, for $k=4$, is shown in Figure 8 and is obtained as follows. We start placing guards on B_1. Three of its walls (left, top and right) are not visible by any potential guard placed on B_i $\ ,\forall i > 1$ (property 1). We place three guards to cover these walls. Consider the space $\wp_1$ between B_1 and B_2. We need two guards to cover $\wp_1$; let one of these guards be placed on B_1 and the other on B_2 (see Figure 8). After placing these two guards, all walls of B_1 are visible, and two walls of B_2 (top and right) are visible. We need an additional guard to cover the left wall of B_2, and this guard does not cover any other wall in the city. Consider now the space $\wp_2$ between the hole B_2 and B_3 and place two guards to cover $\wp_2$; let one of these guards be placed

on B_2 and the other on B_3. After placing these two guards, all walls of B_1 and B_2 are guarded, and two walls of B_3 are guarded. We placed four guards on B_1 and three guards on B_2. We continue this process, and three guards are required to guard each building $B_i \ \forall i > 2$. This results in $3k + 1$ guards as we place three guards on each building $B_i \ , \forall i \in (1, k]$, and four guards on B_1. Guard locations and directions are shown in Figure 8.

Blind Voronoi Game

Omid Gheibi* Hamid Zarrabi-Zadeh†

Abstract

In the classical Voronoi game, two players compete over a user space by placing their facilities in a certain order, each trying to maximize the number of users served by their facilities. We introduce a new variant of the game, so called *blind Voronoi game*, in which the distribution of users in the underlying space is initially unknown to the players. As the game proceeds, players obtain a partial information about the distribution, which is limited to the distance of closest users to the facilities placed so far. We investigate mixed strategies for the players in this blind setting. In particular, we show that in the two-round blind Voronoi game on a line segment, there is a mixed strategy for the second player that guarantees him at least 1/3 of users in expectation. In two dimensions, we show that our strategy guarantees an expected payoff of 1/5 for the second player, when users are arbitrary distributed in the plane. We generalize our strategy to any d-dimensions and to any number of k rounds, for $k, d \geq 1$.

1 Introduction

The Voronoi game was introduced by Ahn *et al.* [1] as a competitive facility location problem. The game consists of two players P1 and P2, and a set of users in an underlying space. Th players compete over the users by alternatively placing their facilities in the space, each trying to maximize the number of users served by their facilities, assuming that each user seeks service from the closest facility.

The Voronoi game has been the subject of active research over the past decade, and various variants of the problem have been studied in the literature, such as Voronoi game on line segments [3], in the plane [10], on graphs [2, 11, 12], and in simple polygons [6]. The problem has been also studied both in one round [7, 9], where P1 places all his facilities before P2 starts placing his facilities, and in the k-round version, where players alternate by placing one facility at a time for k rounds [4]. The problem has been also studied in the discrete and continues spaces. In the former, users are considered as discrete points in the space (e.g., in [3, 4, 5]), while in the latter, users are distributed continuously over the space (e.g., in [1, 7, 10]).

In this paper, we introduce a new variant of the Voronoi game which we call the *blind Voronoi game*. The blind version differs from the previous variants of the game in that the distribution of users is initially unknown to the players. During the game, players obtain a partial information about the distribution, which is limited to the distance of closest users to the facilities placed so far. The blind version of the game is suitable in real situations of the facility location problem, in which no prior information about the location or distribution of the users is available to the players. In other words, players are "blind" to the location and distribution of the users in the space.

We study the *k-round discrete blind Voronoi game*, in which two players P1 and P2 compete over a user space by placing one facility at a time, for a total of k rounds. The user distribution is initially unknown to the players, and the only information available to each player during the game is the distance of closest users to the current facilities. The *payoff* of each player is defined as the fraction of users served by his facilities. The goal of P2 is to maximize his expected payoff in the worst case, where the expectation is taken over the random choices of the player, and the worst case is taken over all possible user distributions. On the other hand, P1's goal is to minimize P2's expected payoff. We note that maximizing payoff for P1 is not a proper goal, as there is always a distribution of users in which the expected number of users for P1 is zero: just consider a distribution very dense at the last facility of P2. Moreover, we note that a pure strategy for the second player will not work, as the player has no information about the location of users, and hence, the adversary can easily adjust a distribution for which the player's payoff is zero. Therefore, in order to guarantee a payoff greater than zero, the second player needs to use a mixed strategy, i.e., use randomization.

We investigate the k-round discrete blind Voronoi game in d-dimensional Euclidean space. In one-dimensional two-round game, where users are discretely distributed on a line segment, we show that the second player has a mixed strategy that guarantees him an expected payoff of 1/3, regardless of the initial distribution of the users. When users are discretely distributed in the plane, our mixed strategy guarantees an expected payoff of 1/5 for the second player in the two-round

*Department of Computer Engineering, Sharif University of Technology, gheibi@ce.sharif.edu

†Department of Computer Engineering, Sharif University of Technology, zarrabi@sharif.edu

game. We generalize our strategy to any d-dimensions and to any number of k rounds, $d, k \geq 1$, yielding an expected payoff of $1/(kd+1)$ for the second player in the worst case.

2 Preliminaries

Let ℓ be a line segment, and π be a distribution of users on ℓ. Given two points $u, x \in \ell$, we say that u is *served* (or is *covered*) by x, if u is closer to x than any other facility on ℓ. Given a point $x \in \ell$ and a distribution π on ℓ, we denote by $N_\pi(x)$ the number of users in the distribution served by x. We assume that no two users or facilities can share the same location. We call two facilities *adjacent* if no user lies on the line segment between them. The *payoff* of each player is defined as the fraction of users served by his facilities. We denote the minimum expected payoff of P2 at the end of the game by Π_2. The goal of P2 is to maximize Π_2, while P1's goal is to minimize Π_2.

A *candidate set* C is a set of points along with a probability function p that assigns to each point $x \in C$ a probability $p(x)$ such that $\sum_{x \in C} p(x) = 1$. We denote by $G(C)$ the minimum expected number of users served by C, i.e., $G(C) = \min_\pi \left\{\sum_{x \in C} p(x) N_\pi(x)\right\}$, where the minimum is taken over all possible distributions π of the users. Each point in a candidate set is called a *candidate point*. A candidate set C with a maximum possible $G(C)$ is called an *optimal candidate set*.

3 Blind Voronoi game on a line segment

In this section, we consider the two-round blind Voronoi game on a line segment. Let ℓ be the underlying line segment. We solve the game in a top-down approach, analogous to the backward induction in extensive games. Namely, we first suppose that all moves are done, except the last move of P2. After resolving the last move of P2 , we suppose that the first moves of P1 and P2 are finished, and resolve the second move of P1. Similarly, we resolve the first move of P2, assuming that P1 has placed his first facility, and finally we consider the first move of P1.

We start by the last move of P2. Depending on the order of facilities on the line and their adjacency, several cases can arise as depicted in Figure 1. In this figure, f and s denote the facilities of the first and the second player, respectively. Moreover, the empty circles show candidate points for the second facility of P2. The cases are distinguished as follows. We assume, w.l.o.g., that the leftmost facility on the line segment is f. (The other case is symmetric.) Thus the first three facilities of the players have two possible permutations: either ffs or fsf. Cases (a) to (d) of Figure 1 correspond to different arrangements for ffs, based on whether two facilities

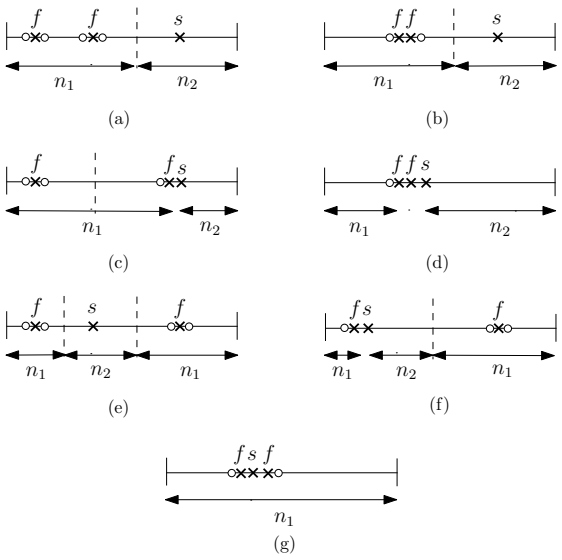


Figure 1: Seven cases for the last move of P2

are adjacent or not. (Adjacent facilities are shown close to each other with no empty circle in between.) Cases (e) to (g) of the figure demonstrate different arrangements for fsf. Any other configuration is analogous or symmetric to one of these cases.

As illustrated in the figure, all candidate points for the second facility of P2 are chosen adjacent to P1's facilities. The next lemma shows that this is indeed the best choice for the candidate sets.

Lemma 1 *Any optimal candidate set for the last move of P2 consists of candidate points adjacent to P1's facilities, on those sides which are free from adjacent facilities. The probability of choosing each of these candidate points in an optimal candidate set must be equal.*

Proof. Let C be an optimal candidate set for P2's last move. Let $\rho(y)$ denote the probability that a point $y \in \ell$ is covered by C, i.e., the sum of probabilities of candidate points that cover y. Note that

$$\begin{aligned} G(C) &= \min_\pi \left\{ \sum_{x \in C} p(x) N_\pi(x) \right\} \\ &= \min_\pi \left\{ \sum_{u \in \pi} \rho(u) \right\} \\ &\geq n \cdot \min_{y \in \ell} \{\rho(y)\}, \end{aligned}$$

where π is taken over all possible distributions of users on ℓ, and y is taken over all points in ℓ where users may be located. Since there is a distribution π where all users are dense at a point y that minimizes $\rho(y)$, we have $G(C) = n \cdot \min_{y \in \ell} \{\rho(y)\}$. This means that

we must cover every point $y \in \ell$ with some positive probability, otherwise $G(C) = 0$. The only way to cover a point y adjacent to a P1 facility is with an adjacent candidate, and therefore, $\rho(y) = p(y)$ for such points. Moreover, the set of adjacent candidates covers all of ℓ. Thus, the minimum of $\rho(y)$ is attained at an adjacent point.

It follows that the optimal strategy is to maximize the minimum of $\rho(y) = p(y)$ at adjacent points. This is achieved by selecting uniformly from just the adjacent points. If one adjacent point is more likely than another, its probability can be reduced and the minimum raised. If a non-adjacent point is in the candidate set C, it can be removed and its probability distributed evenly among the adjacent points, also raising the minimum of $\rho(y)$. □

Lemma 2 *For cases (a) to (g) of Figure 1, the value of* Π_2 *is* $\frac{1}{4}$, $\frac{1}{2}$, $\frac{1}{3}$, 1, $\frac{1}{4}$, $\frac{1}{3}$, *and* $\frac{1}{2}$, *respectively.*

Proof. Fix one of the cases in Figure 1. Let C be the set of candidate points in that case, and k be the number of candidate points in C. Let I be the portion of line segment ℓ not already covered by P2, and n_1 be the number of users lying in I (see Figure 1). The candidate points in C are chosen in such a way that they jointly cover the whole area of I. As the probability of choosing each candidate point is equal by Lemma 1, the expected number of users served by the candidate points in C is n_1/k. Considering that the number of users already served by P2 is $n_2 = n - n_1 \geq 0$, the expected number of users served after the last move of P2 is $n_2 + (n - n_2)/k \geq n/k$. Since n_2 can be zero in a distribution, the above inequality reduces to an equality at its minimum. Therefore, $\Pi_2 = \frac{1}{k}$ in the corresponding case. Since the size of the optimal candidate sets in cases (a) to (g) are 4, 2, 3, 1, 4, 3, 2 respectively, the statement of the lemma follows. □

By Lemma 2, the minimum value of Π_2 achievable by the second player is 1/4. However, this minimum value corresponds to two cases that the second player can avoid by choosing a proper strategy for placing his first facility. We show this in the next lemma.

Lemma 3 *The best move for the first facility of P2 is to place his facility adjacent to the first facility of P1.*

Proof. If the first move of P2 is not adjacent to the first move of P1 (see Figure 2a), then P1 can place his last facility in the second round in such a way that either case (a) or case (e) of Figure 1 occurs, for which we already know that Π_2 is 1/4 by Lemma 2. On the other hand, if P2 places his first facility adjacent to the first facility of P1, he can guarantee a value of $\Pi_2 \geq 1/3$ in all cases derived from his move. □

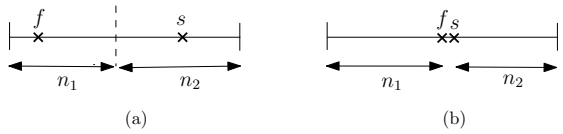


Figure 2: Two cases for the first move of P2

Lemmas 1–3 together yield the following theorem.

Theorem 4 *There is a strategy for the second player in the two-round blind Voronoi game on a line segment that guarantees* $\Pi_2 \geq \frac{1}{3}$.

Remark. It is easy to see that the first player has always a strategy to force Π_2 to be at most 1/3. The first move of P1 is arbitrary, as there is still no other facility, and there is no information about the users. If P2 places his first facility adjacent to the first facility of P1, which is indeed the best strategy for P2 by Lemma 3, then P1 in his second move, can put his second facility far from the first two facilities, in order to either case (c) or case (f) of Figure 1 arises, in both of which Π_2 is 1/3.

4 Blind Voronoi game in $\mathbb{R}^d$

In this section, we generalize our result for the blind Voronoi game in one dimension to any fixed dimension $d \geq 2$, and for any number of rounds $k \geq 2$. In the following, we denote by $r(f)$ the distance of a facility f to its nearest user.

Recall that the main idea behind our strategy in one dimension was to cover all points on the line where users may be located by a set of candidate points. We generalize this idea to d dimensions as follows. Let F be a set of facilities placed by the first player in $\mathbb{R}^d$. We call a set C of points in $\mathbb{R}^d$ a *proper candidate set* with respect to F, if in the Voronoi diagram of $C \cup F$, the unoin of Voronoi cells of C cover all regions in $\mathbb{R}^d$ that may contain users. This ensures P2 to cover every point that a user may be located with some positive probability, when choosing at random from the candidate set.

To present the generalized idea, we start by explaining our strategy in two dimensions. Suppose that P1 has placed his first facility at a point f in the plane. Consider a circle B centered at f with radius $r(f)/2$. We choose three points evenly spaced on the boundary of B as our candidate set C. (See Figure 3.) Note that the Voronoi cell of f in the Voronoi diagram of $C \cup \{f\}$ is contained in B, and hence, is empty of any user. Therefore, all users are covered by the union of the Voronoi cells of C, and hence, C is a proper candidate set. Now, if P2 chooses uniformly at random from C, he receives one third of the users in expectation in the worst case.

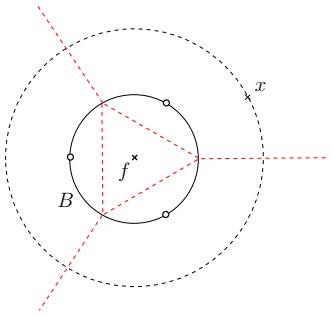


Figure 3: A facility f and its closest user x. Three candidate points are evenly spaced on the boundary of a circle B centered at f of radius $\|x - f\|/2$.

Now, suppose that P1 places his second facility on a point f'. Again, we choose three points evenly spaced on a circle centered at f' with radius $r(f')/2$, and add these three points to C. Now, C has six candidate points, one of which is already chosen by P2 in the first round. (See Figure 4.) Moreover, C is a proper candidate set with respect to $\{f, f'\}$, as the Voronoi cells of f and f' are both empty of any user. Now, P2 chooses from the remaining 5 candidate points of C uniformly at random, and hence, receives at least $1/5$ of the users in expectation. This yields a strategy for the second player in the two-round blind Voronoi game in the plane that guarantees $\Pi_2 \geq \frac{1}{5}$.

To extend our strategy to higher dimensions, we first define some notions. Given a point $p \in \mathbb{R}^d$ and a real value $r > 0$, we denote by $B(p, r)$ a d-dimensional ball of radius r centered at p. The distance of a point p to a point set S is defined as $\min_{q \in S} \|p - q\|$. A straightforward extension of our two-dimensional idea to higher dimensions would be as follows. Let f be a facility of P1, and let $r = r(f)/2$. We choose $d + 1$ points evenly spaced on the boundary of $B(f, r)$ as the candidate points. However, in $d \geq 5$ dimensions, the Voronoi cell of f is no longer contained in $B(f, r(f))$, and hence, there may be users outside $B(f, r(f))$ covered by f. To overcome this issue, we need to choose the radius r small

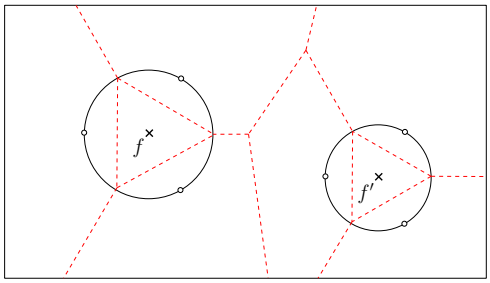


Figure 4: The Voronoi diagram of $C \cup \{f, f'\}$

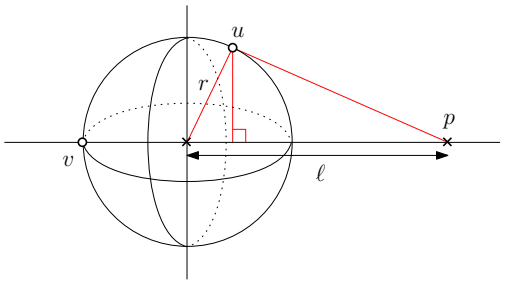


Figure 5: Point p at distance ℓ to the center of a regular simplex.

enough to make sure that the Voronoi cell of f is empty of any user. The following lemma is crucial for finding such a proper radius.

Lemma 5 *Let S be a regular d-dimensional simplex whose circumsphere has radius r, and let p be a point at distance ℓ to the center of S. Then the distance of p to the vertex set of S is at most*

$$\sqrt{r^2 + \ell^2 - \frac{2r\ell}{d}}.$$

Proof. Fix a vertex v of S. Suppose, w.l.o.g., that S is centered at the origin, and that $v = (-r, 0, 0, \ldots, 0)$. As the angle subtended by any two vertices of S through the origin is $\arccos(-1/d)$ [8], the x-coordinate of any other vertex of S is r/d. Now, consider a point p in the halfpence $x \geq 0$ at distance ℓ to the origin. Due to symmetry of S, the maximum distance of p to the vertex set of S is attained when p lies on the line through v and the origin, i.e., $p = (\ell, 0, 0, \ldots, 0)$. Now, fix an arbitrary vertex u of $S \setminus \{v\}$. We can assume w.l.o.g. (by rotating around the x-axis) that u lies in the xy-plane. As u has distance r to the origin, the y-coordinate of u is $\sqrt{r^2 - (r/d)^2}$. (See Figure 5.) Therefore, the distance of p to u (and to any other vertex of $S \setminus \{v\}$) is

$$\sqrt{\left(\ell - \frac{r}{d}\right)^2 + r^2 - \left(\frac{r}{d}\right)^2} = \sqrt{r^2 + \ell^2 - \frac{2r\ell}{d}}.$$

□

Lemma 5 is interesting on its own. For example, it implies that any point on the circumsphere of a regular simplex S has distance at most $\sqrt{(2 - 2/d)}r$ to the vertex set of S, where r is the radius of the circumsphere of S.

We are now ready to provide a general strategy for the blind Voronoi game in any fixed dimensions. In the following we denote by $\mathrm{VD}(P)$ the Voronoi diagram of a point set P.

Theorem 6 *For all integers $k, d \geq 1$, there is a strategy for the second player in the k-round blind Voronoi game in $\mathbb{R}^d$ that guarantees $\Pi_2 \geq \frac{1}{kd+1}$.*

Proof. Suppose we are in the k-th round, $k \geq 1$, and let F be the set of k facilities placed by $P1$. For each facility $f \in F$, we define C_f as a set of $d+1$ points evenly spaced on the boundary of a ball of radius $r(f)/d$ centered at f. We take $C = \bigcup_{f \in F} C_f$ as the candidate set for P2. We claim that C is a proper candidate set. To prove this, we show that the Voronoi cell of every point of F in $\mathrm{VD}(C \cup F)$ is empty of any user. Fix a facility $f \in F$, and let $r = r(f)/d$. Consider an arbitrary user u, and let ℓ be the distance of u to f. Note that $\ell \geq r(f) = rd$. Now, by Lemma 5, the distance of u to the point set C_f is at most $\sqrt{\ell^2 + r(r - \frac{2\ell}{d})}$, which is strictly less than ℓ for all values $0 < r < \frac{2\ell}{d}$. The latter inequality holds, since $r \leq \ell/d$ by our choice of r, and therefore, u is closer to a point in C_f than to f. Hence, u cannot lie in the Voronoi cell of f in $\mathrm{VD}(C \cup F)$, which completes the proof of the claim.

Now, the generalized strategy for the second player is as follows. During the first $k-1$ rounds, P2 places $k-1$ facilities arbitrarily on $k-1$ candidate points of C. In the k-th round, P2 places his k-th facility uniformly at random on one of the remaining candidate points of C. Let S be the final set of facilities placed by P2. Since $S \subseteq C$, for any facility $s \in S$, the Voronoi cell of s in $\mathrm{VD}(C \cup F)$ is completely contained in the Voronoi cell of s in $\mathrm{VD}(S \cup F)$. In other words, any user lying in the Voronoi cell of s in $\mathrm{VD}(C \cup F)$ is covered by s at the end of the game. Let $R \subset S$ be the $k-1$ facilities placed by P2 before the last round, and let n_1 be the number of users lying in the Voronoi cells of R in $\mathrm{VD}(C \cup F)$. As the Voronoi cells of F in $\mathrm{VD}(C \cup F)$ are empty, the remaining $n - n_1$ users are covered by the Voronoi cells of the candidate points in $C \setminus R$. Since the probability of choosing each candidate point in $C \setminus R$ is equal in the last round, the expected number of users lying in the Voronoi cell of the selected facility in the last round is $(n - n_1)/(kd + 1)$, where $kd + 1$ is the size of $C \setminus R$. Therefore, the expected number of users lying in the Voronoi cells of S in $\mathrm{VD}(C \cup F)$ is $n_1 + (n - n_1)/(kd+1) \geq n/(kd + 1)$, which completes the proof. □

5 Conclusion

In this paper, we introduced a new variant of the Voronoi game, so called blind Voronoi game, in which the distribution of users is initially unknown to the players. We provided a mixed strategy for the second player in the two-round blind Voronoi game that guarantees an expected payoff of $1/3$, when users are distributed on a line, and an expected payoff of $1/(2d + 1)$, when users are arbitrary distributed in $\mathbb{R}^d$. Our strategy can be applied to other variants of the problem, such as the k-round m-facility game, where each player places m facilities at each of the k rounds. Other variants of the problem remain open for further research, such as blind Voronoi game on graphs and blind Voronoi game in simple polygons.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments.

References

[1] H.-K. Ahn, S.-W. Cheng, O. Cheong, M. Golin, and R. van Oostrum. Competitive facility location: the Voronoi game. *Theoret. Comput. Sci.*, 310(1):457–467, 2004.

[2] S. Bandyapadhyay, A. Banik, S. Das, and H. Sarkar. Voronoi game on graphs. *Theoret. Comput. Sci.*, 562:270–282, 2015.

[3] A. Banik, B. B. Bhattacharya, and S. Das. Optimal strategies for the one-round discrete Voronoi game on a line. *J. Comb. Optim.*, 26(4):655–669, 2013.

[4] A. Banik, B. B. Bhattacharya, S. Das, and S. Das. Two-round discrete Voronoi game along a line. In *Proc. 3rd Internat. Workshop Frontiers Algorithmics*, pages 210–220, 2013.

[5] A. Banik, B. B. Bhattacharya, S. Das, and S. Das. The 1-dimensional discrete Voronoi game. *Operations Research Letters*, 47(2):115–121, 2019.

[6] A. Banik, S. Das, A. Maheshwari, and M. Smid. The discrete Voronoi game in a simple polygon. *Theoretical Computer Science*, 2019.

[7] O. Cheong, S. Har-Peled, N. Linial, and J. Matousek. The one-round Voronoi game. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 97–101, 2002.

[8] H. S. M. Coxeter. *Regular polytopes*. Dover, New York, 1973.

[9] M. de Berg, S. Kisfaludi-Bak, and M. Mehr. On one-round discrete Voronoi games. In *Proc. 30th Annu. Internat. Sympos. Algorithms Comput.*, pages 37:1–37:17, 2019.

[10] S. P. Fekete and H. Meijer. The one-round Voronoi game replayed. *Comput. Geom. Theory Appl.*, 30(2):81–94, 2005.

[11] M. Mavronicolas, B. Monien, V. G. Papadopoulou, and F. Schoppmann. Voronoi games on cycle graphs. In *Proc. 33rd Internat. Sympos. Math. Found. Comput. Sci.*, pages 503–514, 2008.

[12] X. Sun, Y. Sun, Z. Xia, and J. Zhang. The one-round multi-player discrete Voronoi game on grids and trees. In *International Computing and Combinatorics Conference*, pages 529–540. Springer, 2019.

Line Segment Visibility: Theoretical and Experimental Results

Jonathan Lenchner * Eli Packer †

Abstract

We study a family of line segment visibility problems related to classical art gallery problems, which are motivated by monitoring requirements in commercial data centers. Given a collection of non-overlapping line segments in the interior of a rectangle and a requirement to monitor the segments from one side or the other we examine the problem of finding a minimal set of point guards.

Although we give a handful of new theoretical results, the focus of this paper is the development of new heuristics and to report on our experimental results.

1 Introduction

The problems in this paper arise from the need to cost effectively monitor cooling and heat production in commercial data centers. We model a data center as a two-dimensional rectangular enclosure where the objects of interest from a monitoring perspective are the air intake regions of servers that are housed within racks. The monitoring technique of interest is to use statically placed thermal imaging cameras. Thermal imaging cameras are relatively expensive and so the challenge is to deploy as few of these cameras as possible while still viewing, in total, all of the air intake regions.

In traditional art gallery problems an entire polygonal region must be kept under surveillance. In our case a prescribed collection of non-overlapping line segments in the interior of a *rectangle* must be guarded, and typically it is important just to see one side of each segment.

In this work we focus on pragmatic aspects of these line segment guarding problems. We introduce a new family of problems that we call "all-but-δ" coverage where we allow ourselves to omit some arbitrarily small δ along each segment. In various extremal cases that we shall encounter one or more cameras can be employed to see all points along a collection of segments except for an arbitrarily small length along one of the segments. If we insist on seeing every point along all the segments we must waste a camera just to see this arbitrarily small sub-segment. To make the problem a bit more practical we thus consider the additional covering problem where we admit this all but an arbitrarily small amount, δ, of leeway along all segments. We mention, but do not prove, a set of combinatorial bounds giving how many guards are needed to cover n segments in the worse case, noting how full coverage and "all-but-δ" coverage differ in a series of examples. We then show that even in the simplest guarding models the problem of finding a minimal guard set is NP-Complete. Lastly, we provide heuristics for finding guard sets along with some experiments to evaluate the heuristics in randomly generated data centers.

*IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA; lenchner@us.ibm.com

†Intel Corporation, Tel Aviv 6971011, Israel; eli.packer@intel.com.

2 Related Work

Our work is closely related to classical art gallery problems (see [7], [9], [10] and [13]). In our case a prescribed collection of n non-overlapping line segments in the interior of a rectangle must be guarded, and moreover, typically it is required to see each segment from one particular side. Czyzowicz et al. [4], Toth [11] and Urrutia [13] studied the problem without the presence of a boundary, and where line segment visibility could be from either side – a variant of our problem family that we call the "Solver's Choice" problem – establishing various combinatorial bounds.

The art gallery problem was first posed in 1973 by Klee [9] when he asked how many guards are sufficient to guard the interior of a simple polygon. In 1986 Lee and Lin proved that finding the minimum number of guards for an arbitrary simple polygon is NP-hard [7]. Over the years numerous variants of the art gallery problem have been presented and studied. For the art gallery problem variants closest to the current work [4, 11, 13], NP-hardness has never been established.

For over three decades interest in art gallery problems mainly focused on theoretical aspects. However, since the experimental work of Amit et al. [3] in 2010, a number of experimental projects have been reported, most of which provide approaches to the classical problem variant. The aim of this recent body of experimental work has been to find close-to-optimal solutions in reasonable time. We refer the reader to a survey of some of these projects [6].

3 Problem Setup

We make the following assumptions: segments are assumed to be non-overlapping/non-intersecting and may

not touch the rectangular boundary. A (point) guard's line of site stops when it intersects a segment. A guard collinear with a line segment does not see any point on either side of the segment. Guards may be placed anywhere in the interior of the rectangle but *not* on a line segment.

We consider cases where the segments to be monitored are either all vertical or, failing that, all axis-aligned. These cases are the two most typical cases found in modern data centers. Segments of arbitrary orientation have also been studied [11, 13] though we have not. Within these cases we identify several variants of the basic visibility problem. If visibility must be from a given side, but that side is specified by the problem poser, we say the problem is an instance of the **Poser's Choice** problem. These problems correspond to the case where one is given a data center and it must monitored, as is, as cost effectively as possible. We consider two cases of the Poser's Choice problem, one in which the Poser must be consistent and always request the same side of segments to be guarded (say the top side for horizontal segments, and the left side of vertical segments, in the axis-aligned case), or a second model where the problem poser can require either side of any segment be guarded.

If, on the other hand, the solver has the choice of which side to monitor the segments from, we say that it is an instance of the **Solver's Choice** problem. This model corresponds to the case where one can alter the orientation of the equipment (which side the air intakes face) in order to make monitoring more cost-effective. We also consider two variants of the Solver's Choice problem: a variant where the solver must monitor the entire segment, but may monitor some points from one side and some points from the other side, and a variant where the solver must monitor the entire segment from one side. A final variant is where the solver must monitor the entire segment from both sides.

We use the terms "guards" and "cameras" interchangeably. We decided to constrain our attention to a bounding rectangle rather than a more general polygon because most data centers are rectangular or very nearly rectangular. It also allows us to align with prior work, since in the case of combinatorial bounds (the only aspect of these problems that have been considered by other authors), the presence of a rectangular boundary is equivalent to there being no boundary at all.

4 Combinatorial Bounds

Table 4(a) collects combinatorial bounds from [4], but mostly our own, for all of these problem variants. We recommend the reader refer to this figure as we proceed. Given space constraints and the focus of this paper we shall not provide proofs for most of these bounds. Some of the bounds are covered in our prior work [5, 8]. However, we shall provide a couple of examples establishing lower bounds and sketch one result because of it relevance in the all-but-δ analogs.

In the all-but-δ variations we are satisfied with guarding all segments except for a length of δ along each segment, for a fixed arbitrarily small $\delta > 0$. Table 4(b) collects combinatorial bounds for these problem variants. In all of these cases we ask what is the worst-case number of guards needed to see a set of N segments.

In all the arguments involving combinatorial bounds we first extend segments so that each endpoint is within some arbitrarily small ϵ of the bounding rectangle or a neighboring segment. By doing so the problems become uniformly harder.

The combinatorial bounds when all segments are vertically aligned are rudimentary and covered in [5]. It is easy to convince oneself that the same bounds exist whether or not we allow ourselves to omit some arbitrarily small δ along each segment. The first case where there is a provably different bound for the case where one must see every point along all segments compared with seeing all but δ is the case of axis aligned segments and the Poser's Choice problem variant. In this case, Figure 1, due to Toth [12], provides an example of

Figure 1: A set of n segments (in black) for the Poser's Choice problem, requiring $\frac{2n}{3}$ cameras. A camera that sees everything within a given "H" but a small amount along the left segment has been placed just above and to the right of the associated horizontal segments.

n segments that require $\frac{2n}{3}$ cameras to see all segment sides entirely. The little red "tick marks" indicate which sides of the segments the camera must guard. It is not difficult to see that two cameras are required to completely see the 3 segments comprising each "H." Using one camera in each "H", e.g., the one's pictured, will necessarily leave some small unseen sub-segment along one of the component vertical segments. Note that $\frac{2n}{3}$ cameras does *not* serve as a lower bound for the all-but-δ variant of Poser's Choice. Indeed, it turns out that we can establish an $\frac{n}{2}$ lower bound in this case. Following Czyzowicz et al. [4], we take any partition of the bounding rectangle derived from an axis-aligned finite set of segments, e.g., as depicted in Figure 2 and think of the result as a set of rectangular "rooms." We then form a graph where the nodes are the rooms and there is an edge between any two rooms that have an

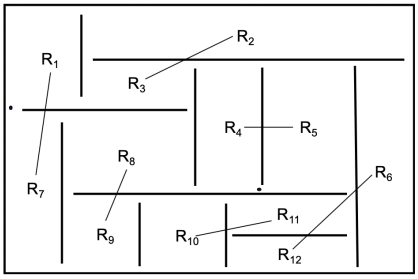


Figure 2: $n = 11$ segments extended so they come within some very small ϵ of one another, or of one of the rectangular walls, leaving $n + 1 = 12$ rectangular "rooms." A matching, as described in the text is shown. Cameras corresponding to the two edges connecting (R_1, R_7) and (R_4, R_5) are depicted, just above and to the left of the segment separating rooms R_1 and R_7, and just below and to the left of the segment separating rooms R_4 and R_5. The first of the cameras sees all of the requisite side of rooms R_1 and R_7, while the second camera must omit some arbitrarily small segment along the top "wall" above rooms R_4 and R_5, if this segment side needs to be guarded.

open "door" between them. An application of Tutte's Theorem enables one to conclude that a near perfect matching exists in this graph. If we are happy to omit an arbitrarily small δ from the walls of these rooms we can ensure our ability to see the needed segment sides in any pair of rooms in the near perfect matching with a single camera, thus establishing the claimed $\frac{n}{2} + O(1)$ lower bound.

A related example is that given in Figure 3 for the axis

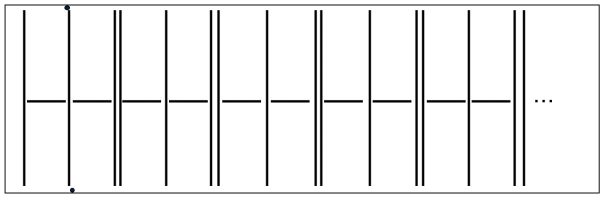


Figure 3: A set of n axis aligned segments following the repeating pattern vertical-horizontal-vertical-horizontal-vertical, beginning on the left. The set requires $\lceil \frac{2n}{5} \rceil$ cameras to entirely see all segments from the solver's choice of sides assuming one must see a segment entirely from one side. Two cameras sufficient to see the first 5 segments are shown.

aligned Solver's Choice variant in the case where one must see a segment entirely from one side or another. Starting from the left, we have a repeating sequence of vertical-horizontal-vertical-horizontal-vertical segments (i.e. VHVHV, VHVHV, ...), all segments stretching to within a very small ϵ of the bounding rectangle or the closest adjacent segments. A simple analysis shows that in order to see the left three segments in this figure entirely without leaving a small unseen subsegment on either of the first two vertical segments one must position two cameras each no further right than just fractionally beyond the second vertical segment – such a set of two cameras is depicted in the figure. Given that the vertical segments extend to within ϵ of the bounding rectangle, these two cameras see a negligible amount of any segment beyond the 5th segment from the left. But then, starting at the 6th segment, we have the same visibility problem we started out with, and thus conclude that this arrangement of segments requires the stated $\lceil \frac{2n}{5} \rceil$ cameras to see everything.

Note, however, that if it were sufficient to see all but some arbitrarily small length δ of each segment, then we could actually get away with one camera for every three segments in this example. The segments may be thought of as a sequence of triplets of segments. Using the previous notation, they cycle through the sequence VHV, HVV, HVH, VVH and then back to VHV. One may guard all but δ of each VHV triplet by placing a guard just above and to the right of the horizontal segment, and any of the other triplets can be guarded entirely with a single strategically placed camera. In this all-but δ variant of Solver's Choice we know of no worse case then the one that required $\lceil \frac{n}{3} \rceil$ cameras for all vertical segments.

The tables in Figure 4 describe the state of our knowledge regarding both upper bounds (prefixed by 'U') and lower bounds (prefixed by 'L'), for each of the problems we have examined, up to constant factors. Table 4(a) gives the bounds for the complete coverage problems and Table 4(b) gives the bounds where we require all but some arbitrarily small δ of the segments to be seen.

5 Hardness Results

The proofs in this section are relegated to the Appendix.

Lemma 1 *If we constrain all n segment $s_i = [(a_i, b_i), (a'_i, b'_i)]$ to have rational coordinates, and ℓ denotes the maximum absolute value of any of the numerators, m the maximum value of any of the denominators, and, furthermore, we define the complexity of the problem to be* $\max(\lg \ell, \lg m, n)$*, then all of the problem variants we have considered thus far are in NP. Without the constraint of rational coordinates all problem variants are in NP within the real RAM model of computation.*

Theorem 2 *Given a set of all vertical segments, it is NP-Complete to find a minimum set of guards that see all segments from the left side, where the problem complexity is defined as in Lemma 1.*

Sidedness Constraint / Segment Orientation	Solver's Choice ⫽/⫽	Solver's Choice /	Poser's Choice (All segments from same specified side)	Poser's Choice	Both Sides
Vertical	U n/3 L n/3	U n/3 L n/3	U n/2 L n/2	U n/2 L n/2	U 2n/3 L 2n/3
Orthogonal	U n/2 [4] L n/3	U n/2 L 2n/5	U n/2 L n/2	U 3n/4 L 2n/3 [12]	U 4n/5 [11] L 2n/3

(a) Complete Coverage

Sidedness Constraint / Segment Orientation	Solver's Choice ⫽/⫽	Solver's Choice /	Poser's Choice (All segments from same specified side)	Poser's Choice	Both Sides
Vertical	U n/3 L n/3	U n/3 L n/3	U n/2 L n/2	U n/2 L n/2	U 2n/3 L 2n/3
Orthogonal	U n/2 L n/3	U n/2 L n/3	U n/2 L n/2	U n/2 L n/2	U 4n/5 L 2n/3

(b) All-but-δ Coverage

Figure 4: What we know about the various combinatorial bounds. U denotes the upper bound, L the lower bound. All results are modulo additive constants. Citations to results that are not our own are given in square brackets.

Theorem 3 *The following variants are NP-hard:* **(a)** *Finding a minimum set of guards for vertical segments where the poser chooses the sides to be guarded.* **(b)** *Finding a minimum set of guards for vertical segments where both sides are to be guarded.* **(c)** *Finding a minimum set of guards for segments with any orientation where the poser chooses the sides to be guarded.* **(d)** *All the variants stated in this theorem and the preceding one where the visibility region of each guard is restricted to lie within a cone of some given angle emanating from the guard.* **(e)** *Same as Theorem 2 where at least one point on each segment has to be guarded.* **(f)** *Finding a minimum set of guards for vertical segments where the solver chooses the sides to be guarded.*

The proofs of all these variations involve just small changes to the proof of Theorem 2 and are therefore omitted. We note that Theorems 2 and 3 also hold for the all-but-δ case, as we describe in the Appendix.

6 Heuristics: Mixed Integer Programming

We used a Mixed Integer Programming (MIP) tool to solve guarding problems, not for segment sides, but for a closely related problem that is more faithful to the geometry of a computer data center. In these problems we are given a set of rectangles inside a bounding rectangle and for each rectangle we are required to guard the Poser's Choice of sides from the outside. We shall later consider the Solver's Choice variant of this problem.

The main idea is to place guard candidates (denoted by G) within the cells of a grid and among these candidate locations, select a smallest set that guards all required edges (denoted by E). To increase the guarding quality, as a second priority we prefer guards that are both near edges and see the edges from wider angles (both properties mean better sight resolution). To evaluate the guarding quality, we proceed as follows. For any edge $e \in E$ seen by a candidate guard $g_i \in G$, let e_c be the center of e and v_e be a unit vector, perpendicular to e and directed inside the rectangle of e. Let v_{ei} be a unit vector directed from g_i to e_c. Let d_{ei} be the distance from e_c to g_i and let $a_{ei} = 1 - v_e \cdot v_{ei}$ where the operator $\cdot$ represents dot product. a_{ei} then measures the viewing angle from g_i. Then $q_{ei} = d_{ei} + ra_{ei}$ measures the viewing quality where $r > 0$ balances between the distance and the angle measures. Let $q_i = \frac{\Sigma_{e \in E_i} q_{ei}}{|E_i|}$ where E_i is the set of edges visible from g_i. As q_i is a function of distances and viewing angles with respect to all visible edges, it will serve as the candidate's quality measure. Let $Q = max_{1 \le i \le m}(q_i)$, where m is the number of candidates, be the largest such measure over all candidates. As we need to make sure that the solution contains a minimum set of guards regardless of the guarding quality, we penalize each guard i by $(Q|G|+q_i)$. By using this penalty, it is easy to verify that a minimum set of guards is generated and as a second priority, candidates with better guarding qualities are preferred.

Let $M_{n \times m}$ be a matrix whose entry $m_{i,j}$ is 1 if and only if candidate $g_j \in G$ guards edge $e_i \in E$ by the above definition. Let $\vec{C}$ be the decision variables of our model. It is a vector of size m with i-th element C_i corresponding to whether $g_i \in G$ is chosen ($C_i = 1$) or not ($C_i = 0$). Then the formulation as an IP instance is:

Minimize: $\Sigma_{i \in \{1 \ldots m\}}((q_i + Q|G|)C_i)$
Subject to: $M \cdot \vec{C} \ge \vec{1}, \quad \vec{C} \in \{0,1\}^m$

where $\cdot$ is the matrix multiplication operator.

We start with a sparse grid, and incrementally execute the MIP with denser and denser grids until the solution stabilizes. We define stabilization as a sequence of MIP executions where no improvement of the best solution occurs. We denote the length of this sequence by STABLE_ITERS and use it as a program parameter.

When the instance is too large, there is a risk that the MIP will not complete its execution within a rea-

sonable amount of time. Hence, we limit the MIP's running time so that when a user-specified wait time, MAX_WAIT, expires, we relax the integer constraints on the guard candidates and let them take arbitrary non negative values (so shifting to Linear Programming, or LP for short). Given the LP results, we find a guarding set by iterating over them in the reverse order of their LP values (so from high to low), adding guards as long as not all required segments are guarded.

A high level description of the algorithm is given in the pseudo-code below. The parameters for the algorithms are:

- racks - racks with associated edges to be guarded
- gs - initial grid size
- gsif - how much to increase the grid density by with each iteration. If we want to increase the grid density by 1% each time, then $gsif = 1.01$
- STABLE_ITERS, MAX_WAIT - defined above

Algorithm 1 Guarding Placement(racks, gs, gsif, STABLE_ITERS, MAX_WAIT)

```
1: i ⇐ 0
2: best ⇐ ∞
3: bestSolution ⇐ NIL
4: solver ⇐ IP
5: while i < STABLE_ITERS do
6:     grid ⇐ build a grid with size gs
7:     Compute the visibility matrix M
8:     solution ⇐ execute solver
9:     if solution found within time MAX_WAIT then
10:        if solution.size < best then
11:            best ⇐ solution.size
12:            bestSolution ⇐ solution
13:            i ⇐ 0
14:        end if
15:        i ⇐ i + 1
16:        gs ⇐ gs ∗ gsif
17:    else
18:        i ⇐ 0
19:        solver ⇐ LP
20:    end if
21: end while
22: return bestSolution
```

In the Solver's Choice guarding model, when we are considering rectangles rather than segments, we designate pairs of opposite sides of each rectangle that should be guarded and the problem solver has the choice of which one to guard. To model this problem we replace the visibility matrix M with a pair of matrices M_L, M_R, where M_L describes what guards see the Left or Top edges of the associated rectangles and M_R describes what guards see the Right or Bottom edges of the associated rectangles. In the IP and LP instances we replace $M \cdot \vec{G} \geq \vec{1}$ with $(M_L + M_R) \cdot \vec{G} \geq \vec{1}$.

There is one catch with this model. When we go through the pre-processing step of breaking segments into sub-segments that are entirely seen by the respective candidate guards, as in the proof of Lemma 1), the solution may end up seeing some sub-segments of a given segment from one side and some from the other. Thus, the model corresponds to the left-most case of Solver's Choice from Figure 4a. At this point we only know how to get our heuristc to work for the Solver's Choice variant where one must see one side or the other entirely if we don't allow shared guarding of segments by multiple cameras. We have not yet run experiments for either of the Solver's Choice visibility models.

7 Experiments

We used the Python MIP package [2] to execute both the MIP and LP. We ran on a 64-bit Windows OS machine with Intel (R) Core(TM) i7-6700 CPUs running at 4.00GHz, configured with 4 cores and 8 logical processors. To increase the speed, we used the Numba package [1] to parallelize the construction of the visibility matrix, M. Taking advantage of the Nvidia GPU card **GeForce GTX 1080** installed on our machine, Numba speeded up the running time by roughly a factor of 500. Using this speedup, a brute force computation of M was fast enough to be negligible compared to the execution time of the solver.

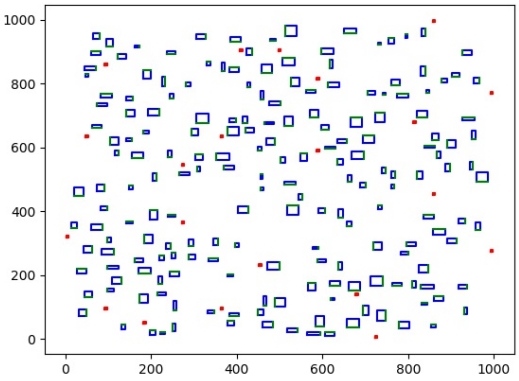
In our experiments we created random, disjoint, axis aligned rectangles inside a bounding box to simulate racks. For each rack we randomly selected one side to be guarded. The racks were laid down on a 1000x1000 unit grid so that each edge was of maximum length 50 units. We started our iterations with candidate guards placed within cells of size 50x50 units.

We carried out experiments with different numbers of racks (10 to 300 with steps of 10). Some illustrative examples with the found guard sets are shown in Figure 5. For each size, we ran several experiments and have reported on the average performance. As we reached as many as 100x100 guarding candidates on a 200 rack set, our IP solver worked on input with 10000 decision variables and 200 constraints. We allowed the IP to run for 1000 seconds before switching to an LP.

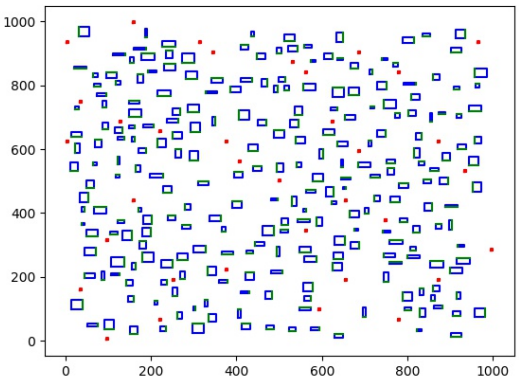
Since the IP was the main bottleneck, we focus here on the time taken to complete, when the heuristic actually came up with a real solution (as opposed to cases where the grid was too sparse to provide a solution). The performance is shown in Figure 6 (note the log scale). When the number of racks exceeded 260, the program switched very early on to an LP and as a result

completed very fast. Hence, these instances are omitted from Figure 6. Using a log scale the exponential growth in time of the MIP solver becomes evident. Nevertheless, we have succeeded in emulating instances that are proportional in size to large real-world data centers.

Figure 7 shows how the number of requisite guards scales with the number of racks. We note an approximately linear relationship, with, on average, about one guard needed for every 10 racks. The slightly increased slope once the number of racks reached 260 may be attributable to the switch to using the rounded LP solver.



(a) 190 Racks



(b) 250 Racks

Figure 5: Data centers with with 190 and 250 randomly generated racks. The racks are drawn in blue, with the sides that need to be guarded given in green. The generated guard set is displayed in red.

8 Conclusions

Motivated by monitoring requirements in data centers we introduced a variety of line segment guarding models including the more practical all-but-δ variant. In addition, we described some new best combinatorial bounds for these problems and showed that all problems are NP-Complete to solve exactly. Our recent focus has been on developing heuristics that scale to the size of large commercial data centers (several hundred thousand square feet or more). One such heuristic places candidate guards within successively finer grids and solves the associated IP until a stable solution is found. When the problem instances get too large and the IP fails to return a solution within a threshold amount of time we resort to LP-rounding. Much work remains, namely to run experiments on the Solver's Choice variants, to provide theoretical guarantees on the performance of our heuristic, as well as to benchmark how well the heuristic does in cases where we know tight bounds from the combinatorial analysis. On the pure theory side there remains for us to tighten the combinatorial bounds and move from segment visibility to the rectangle visibility model.

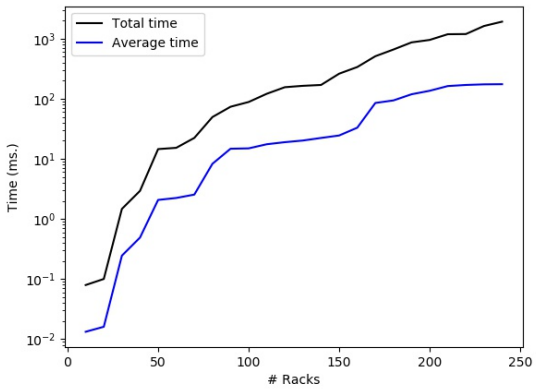


Figure 6: Time taken to run the IP for different numbers of racks.

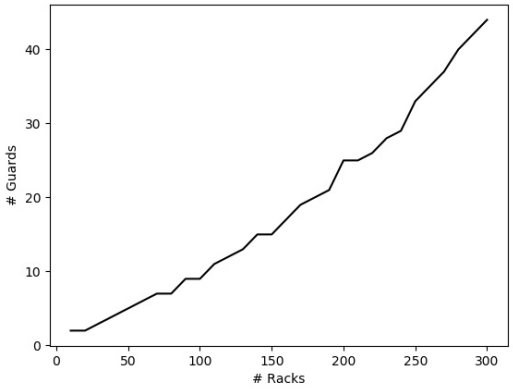


Figure 7: Number of guards for different numbers of racks.

References

[1] *http://numba.pydata.org.*

[2] *https://pypi.org.project/mip.*

[3] Y. Amit, J. S. B. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. *International Journal of Computational Geometry Applications*, 20:601–630, 2010.

[4] J. Czyzowicz, E. Rivera-Campo, J. Urrutia, and J. Zaks. On illuminating line segments in the plane. *Discrete Mathematics*, 137:147–153, 1995.

[5] R. Das, J. Kephart, J. Lenchner, and E. Packer. Internal surveillance problems in data centers. *Fall Workshop on Computational Geometry*, 2008.

[6] P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kroller, and D. C. Tozoni. Engineering art galleries. *CoRR*, abs/1410.8720, 2014.

[7] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory IT-32*, pages 276–282, 1986.

[8] J. Lenchner and E. Packer. Visibility problems concerning one-sided segments. *Fall Workshop on Computational Geometry*, 2012.

[9] J. O'Rourke. *Art gallery theorems and algorithms.* Oxford University Press, Oxford, 1987.

[10] T. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 90(9):1384–1399, 1992.

[11] C. Tóth. Illuminating disjoint line segments in the plane. *Discrete and Computational Geometry*, 30:489–505, 2003.

[12] C. Tóth. Personal communication, 2008.

[13] J. Urrutia. Art gallery and illumination problems. In J. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027. Elsevier Science Publishers, Amsterdam, 2000.

Appendix

Proof of Lemma 1. Given a claimed guard set $\{c_i\}_{i=1}^{M}$, we must simply show that we can verify that the guard set covers the set of segments to be guarded, $S = \{s_i\}_{i=1}^{N}$, in polynomial time. The problem then reduces to verification of coverage of single arbitrary segment from among the set S, say s_1. Let us consider the problem of determining what sub-segments of s_1 are visible from a representative camera we denote by C. WLOG we may assume s_1 is vertically aligned with endpoints $A_1 = (x_1, a_1), B_1 = (x_1, b_1)$ and $b > a$. Further, in the consideration of the effect of segments among $\{s_2, ..., .s_N\}$ that may be blocking C's view of s_1, we only care about the projection of each s_i onto $[A_1, B_1]$ from the vantage point of C. See Figure 8. Thus if $s_2 = [A_2, B_2]$ with $A_2 = (x_2, a_2), B_2 = (x_2', b_2)$ then we care about the projection of A_2 and B_2 onto $[A_1, B_1]$ from c. If only $B_2 \in \triangle(C, A_1, B_1)$, as depicted in Figure 8, then we replace A_1 with B_{2p}, which is the projection of B_2 onto $[A_1, B_1]$. If only $A_2 \in \triangle(C, A_1, B_1)$ then we replace B_1 with the A_{2p}, the projection of A_2 onto $[A_1, B_1]$, while if both $A_2, B_2 \in \triangle(C, A_1, B_1)$ then we must keep track of two sub-segments, $[A_1, B_1'], [A_1', B_1]$, rather than the former one, where now $B_1' < A_1'$ (unless C, A_2, B_2 are collinear).

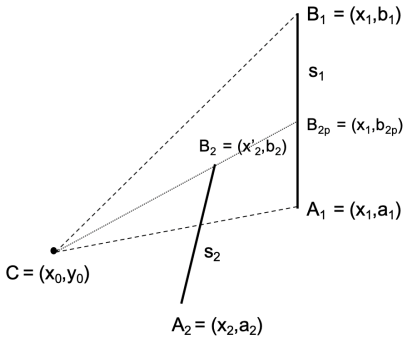


Figure 8: The analysis of camera C's view of segment $s_1 = [A_1, B_1]$, which we assume to be vertically oriented. In considering segment s_2's impact on C's visibility we consider the projection of its endpoint B_2 onto $[A_1, B_1]$ – designated by $B_{2p} = (x_1, b_{2p})$ in the Figure.

The computation of the coordinates of the new endpoints involves just a constant number of additions, subtractions, multiplications and divisions of the numerators and denominators of the prior endpoint coordinates, and each time we consider an additional intervening segment from $[s_2, ..., s_N\}$ we break the original segment s_1 into at most one new sub-segment. In the end, for each camera, we are left with an ordered sequence of at most N visible sub-segments of the original segment, that must be merged with the analogous sequences of sub-segments from each of the other cameras to verify complete coverage of the original segment s_1.

For this purpose, we will sort the y-coordinates of all endpoints, and for each number in the list also keep track of whether it is associated with the bottom (B) endpoint of a segment or the top (T) endpoint of a segment. In the sorted list insure that for endpoints with equal y-values, that the Bs come before the Ts. Finally we walk once through the sorted list of pairs $(y_j, \Gamma_j), \Gamma_j \in \{T, B\}$, treating each B as an open parenthesis, and each T as a close parenthesis. When the number of close parentheses equals the number of open parentheses, say when the pair (y_k, T) is encountered, we check whether there is a gap between y_k and (y_{k+1}, B) in the next pair, assuming that (y_k, T) is not itself the last entry in the sorted list. If any such gap is found, or if, in the sorted list of endpoints, the smallest number does not equal the bottom coordinate of the segment, s_1, or if the largest number does not equal the top coordinate of the segment s_1 then there is a coverage problem. Otherwise the coverage is complete. This sorting and verification process only requires the comparison of two reals or rationals, depending on the problem setup and analogous model of computation, and so the entire verification process can be performed in polynomial time. Thus all of our guarding problems are in NP in their respective models of computation. □

Proof of Theorem 2 We show a reduction from 3-SAT. Let N be the number of variables and M be the number of

clauses. Let $K = max(N, M)$.

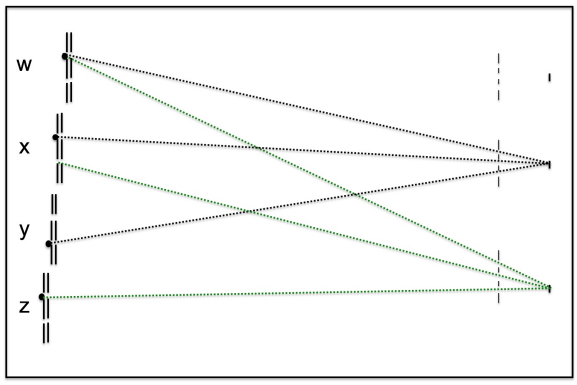


Figure 9: Reduction from 3-SAT: On the left are gadgets for variables w, x, y and z. On the right are gadgets for clauses $(w \vee x \vee \bar{y})$ and $(w \vee \bar{x} \vee z)$. Guards associated with the respective literals are indicated as circles to the left of the respective truth or false gaps. The line of site of a guard to the critical rightmost segment of each clause is indicated with a dotted line. Lines of sight for guards in the same clause are each of the same color.

Figure 9 illustrates the reduction. Each variable gadget consists of a batch of six segments (placed on the left side of the figure) and each clause gadget consists of a batch of five segments (placed on the right side of the figure). The entire structure is placed within a large-enough bounding box.

Each variable gadget consists of two identical columns of three segments, shifted horizontally by a small amount. The length of each segment is $\frac{K-1}{3K}$ and the vertical gaps between the segments are of length $\frac{1}{2K}$. Thus, the vertical length of the entire gadget is 1 unit. The variable gadgets are placed from top to bottom. The vertical distance between the columns of each gadget is set to $\frac{1}{K^3}$. Thus, the vertical length of the entire variable gadget structure is upper bounded by $K + (K-1) * \frac{1}{K^3}$. We also shift the gadgets in a way that the horizontal distance between every two neighboring gadgets is $\frac{1}{K^2}$.

In each variable gadget we refer to the gaps between the top two segments and the middle two segments (from two columns) as the *truth* gap, and the gap between the middle two segments and the bottom two segments as the *false* gap. We refer to either gap as a *literal gap*. As we see later, there is a correspondence between locating guards slightly left of the left gaps and the assignments of the variables.

The clause gadgets are located on the right in a vertical structure. Each gadget consists of one short vertical segment on the right and four vertical segments with the same x-coordinate to its left. Let some clause gadget be denoted by c and let r be the right segment of c, and let L refer to the four segments to the left of r. The length of r is set to $\frac{1}{k^4}$ and its horizontal distance from L is $\frac{K}{1.1K-1}$. The total length of L is one unit and r is placed vertically in the middle of L so that if the vertical coordinate of the bottom of L is y, the bottom coordinate of r will be $y + 0.5 - \frac{1}{2K^4}$.

We define the guarding instance so that all segments have to be guarded from the left. Based on the construction above, the segments L will block r from seeing almost the entire structure of the variable gadgets. The three gaps in between the segments of L constitute the only way that r can *see* the variable gadget structure. They are fine-tuned so that r *sees* the gaps in the variable gadgets that corresponds to the gadget literals (as described above). For that, the three gaps in between the segments of L are located on the line connecting the middle of r to the middle of the corresponding literal defined above. Based on our construction, due to triangle similarities and angle computations, if we place a guard $\frac{1}{K^6}$ units to the left of the left gap of the corresponding litreral, it will see the entire right segment r of the corresponding clause gadget.

For example, consider Figure 9 which corresponds to a formula with three clauses (two of which are given in detail) and four variables (w, x, y and z). Consider the middle clause gadget c' that corresponds to $(w \vee x \vee \bar{y})$: r' *sees* the three literal gaps that correspond to w, x and $\bar{y}$ (the lines of sight from the guards to the critical segment r' are drawn with dashed lines). The third clause, $(w \vee \bar{x} \vee z)$, is similarly depicted.

Next we prove that a 3-SAT formula with N variables is satisfied if and only if the segments are guarded by N guards.
$\Rightarrow$ **Direction:** Suppose the 3-SAT formula is satisfiable. We place one guard near each corresponding literal gap as described above. Next we show that all segments are guarded:

- Since the guard is located to the left of a gap, very close to it near one of the gaps, he guards the segments of its gadget.
- The four left segments of the clause gadgets are guarded by all guards due to our construction and since the guards can see through the variable gadgets.
- Due to our construction and since the 3-SAT formula is satisfied, the right segments of all clause gadgets are guarded by at least one guard.

$\Leftarrow$ **Direction:** Suppose all segments are guarded by N guards. In order to guard the right three segments of each variable gadget, we are forced to place a guard either inside the vertical slab defined by the two columns of each of the N variable gadgets or very close to their left gaps, consuming all the guards. As all the right segments of the clause gadgets are guarded as well, it means that at least one of the guards guard each of the right segments. Thus some or all of the guards are placed in the line of sight of the right segments. As each of the guards is associated with a variable, together with this constraint the value of some or all of the variables are determined so that the formula is satisfied.

We finally note that the reduction is polynomial in the size of the input: each variable and each clause was translated to a constant number of segments. Also, the length of the segments are polynomial in the input, thus the necessary precision is polynomial as well. □

To see that Theorem 2 generalizes to the all-but-δ case, observe that we are alternately hiding and revealing more than a single point on the right-most segments from the guards associated with the respective variable gadgets, and hence we can coordinate hiding/revealing a sufficiently small length δ along all the segments.

Session 4B

Some Geometric Applications of Anti-Chains

Sariel Har-Peled* Mitchell Jones†

Abstract

We present an algorithmic framework for computing anti-chains of maximum size in geometric posets. Specifically, posets in which the entities are geometric objects, where comparability of two entities is implicitly defined but can be efficiently tested. Computing the largest anti-chain in a poset can be done in polynomial time via maximum-matching in a bipartite graph, and this leads to several efficient algorithms for the following problems, each running in (roughly) $O(n^{3/2})$ time:

(A) Computing the largest Pareto-optimal subset of a set of n points in $\mathbb{R}^d$.

(B) Given a set of disks in the plane, computing the largest subset of disks such that no disk contains another. This is quite surprising, as the independent version of this problem is computationally hard.

(C) Given a set of axis-aligned rectangles, computing the largest subset of non-crossing rectangles.

1 Introduction

Partial orderings. Let $(V, \prec)$ be a ***partially ordered set*** (or a poset), where V is a set of entities. An ***anti-chain*** is a subset of elements $D \subseteq V$ such that all pairs of elements in D are incomparable in $(V, \prec)$. A ***chain*** is a subset $C \subseteq V$ such that all pairs of elements in C are comparable. A chain *cover* $\mathcal{C}$ is a collection of chains whose union covers V. Observe that any anti-chain can contain at most one element from any given chain. As such, if $\mathcal{C}$ is the smallest collection of chains covering V, then for any anti-chain D, $|D| \leq |\mathcal{C}|$. Dilworth's Theorem [3] states that the minimum number of chains whose union covers V is equal to the anti-chain of maximum size.

Implicit posets arising from geometric problems. We are interested in implicitly defined posets, where the elements of the poset are geometric objects. In particular, if one can compute the largest anti-chain in these implicit posets, we obtain algorithms solving natural geometric problems. To this end, we describe a framework for computing anti-chains in an implicitly defined poset $(V, \prec)$, under the following two assumptions: (i) comparability of two elements in the poset can be efficiently tested, and (ii) given an element $v \in V$, one can quickly find an element $u \in V$ with $v \prec u$.

As an example, let P be a set of n points in the plane. Form the partial ordering $(P, \prec)$, where $q \prec p$ if p dominates q. One can efficiently test comparability of two points, and given q, can determine if it is dominated by a point p by reducing the problem to an orthogonal range query. Observe that an anti-chain in $(P, \prec)$ corresponds to a collection of points in which no point dominates another. The largest such subset can be computed efficiently by finding the largest anti-chain in $(P, \prec)$, see Lemma 6.

Previous work. Posets have been previously utilized and studied in computational geometry [10, 5, 8]. For the poset $(P, \prec)$ described above, Felsner and Wernisch [5] study the problem of computing the largest subset of points which can be covered by k-antichains.

Our results. We describe a general framework for computing anti-chains in posets defined implicitly, see Theorem 4. As a consequence, we have the following applications:

(A) Largest Pareto-optimal subset. Let $P \subseteq \mathbb{R}^d$ be a set of n points. A point $p \in \mathbb{R}^d$ ***dominates*** a point $q \in \mathbb{R}^d$ if $p \geq q$ coordinate wise. Compute the largest subset of points $S \subseteq P$, so that no point in S dominates any other point in S. In two dimensions this corresponds to computing the longest downward "staircase", which can be done in $O(n \log n)$ time (our algorithm is not interesting in this case). However, for three and higher dimensions, it corresponds to a surface of points that form the largest Pareto-optimal subset of the given point set.

(B) Largest loose subset. Let $\mathcal{D}$ be a set of n regions in $\mathbb{R}^d$. A subset $S \subseteq \mathcal{D}$ is ***loose*** if for every pair $\mathsf{d}_1, \mathsf{d}_2 \in S$, $\mathsf{d}_1 \not\subseteq \mathsf{d}_2$ and $\mathsf{d}_2 \not\subseteq \mathsf{d}_1$. This is a weaker concept than *independence*, which requires that no pair of objects intersect. Surprisingly, computing the largest loose set can be done in polynomial time, as it reduces to finding the largest anti-chain

*Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; sariel@illinois.edu; http://sarielhp.org/.

†Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; mfjones2@illinois.edu; http://mfjones2.web.engr.illinois.edu/.

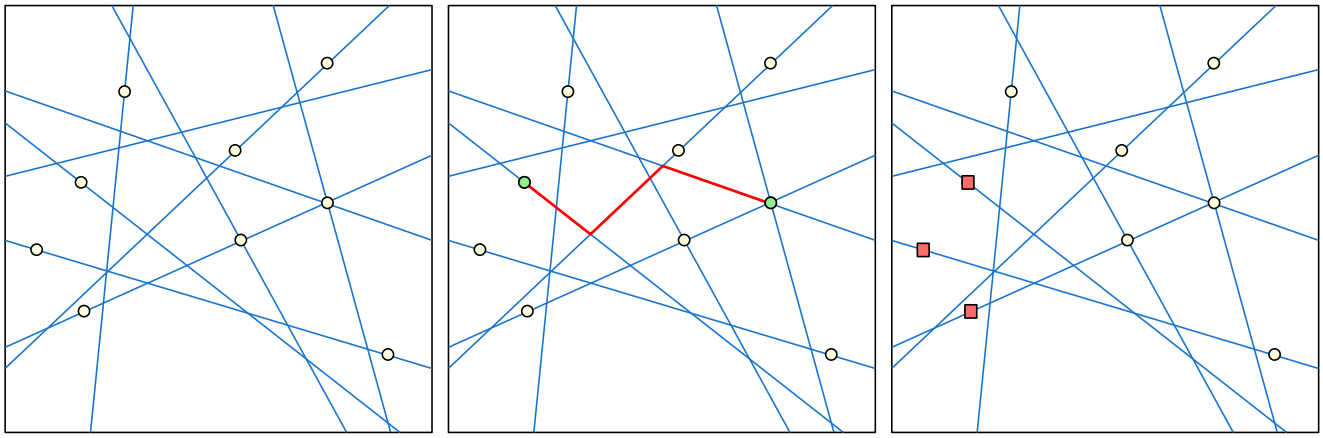


Figure 1.1: Left: A set of points on lines. Middle: A point reaching another. Right: An isolated subset.

Computing largest subset of	Entities	Running time	Ref
Pareto-optimal	Points in $\mathbb{R}^d$, $d > 2$	$\widetilde{O}(n^{1.5})$	Lemma 6
Loose	Arbitrary regions in $\mathbb{R}^d$	$O(n^{2.5})$	Corollary of Lemma 1
	Arbitrary regions in $\mathbb{R}^d$ with a dynamic range searching data structure, $Q(n)$ time per operation	$O(n^{1.5}Q(n))$	Lemma 10
	Disks in the plane	$\widetilde{O}(n^{1.5})$	Corollary 12
Non-crossing	Axis aligned rectangles in $\mathbb{R}^2$	$\widetilde{O}(n^{1.5})$	Lemma 15
Isolated	Points on lines in $\mathbb{R}^2$	$O(n^3)$	Lemma 17

Table 1.1: Our results, where $\widetilde{O}$ hides factors of the form $\log^c n$ (c may depend on d).

in a poset. Compare this to the independent set problem, which is NP-Hard for all natural shapes in the plane (triangles, rectangles, disks, etc).

(C) Largest subset of non-crossing rectangles. Let $\mathcal{R}$ be a set of n axis-aligned rectangles in the plane. Compute the largest subset of rectangles $S \subseteq \mathcal{R}$, such that every pair of rectangles in S intersect at most twice. Equivalently, S is ***non-crossing***, or S forms a collection of pseudo-disks.

(D) Largest isolated subset of points. Let L be a collection lines in the plane (not necessarily in general position), and let P be a set of points lying on the lines of L. A point $p \in P$ can ***reach*** a point $q \in P$ if p can travel from left to right, along the lines of L, to q. A subset of points $Q \subseteq P$ are ***isolated*** if no point in Q can reach any other point in Q using the lines L, see Figure 1.1.

Our results are summarized in Table 1.1.

2 Framework

2.1 Computing anti-chains

The following is a constructive proof of Dilworth's Theorem from the max-flow min-cut Theorem, and is of course well known [9]. We provide a proof for the sake of completeness.

Lemma 1 *Let $(V, \prec)$ be a poset. Assume that comparability of two elements can be checked in $O(1)$ time. Then a maximum size anti-chain in $(V, \prec)$ can be computed in $O(n^{2.5})$ time.*

Proof. Given $(V, \prec)$, construct the bipartite graph $G = (U, E)$, where $U = V^- \cup V^+$ and V^-, V^+ are copies of V. Add an edge (v^-, u^+) to E when $v \prec u$ in $(V, \prec)$. Next, compute the maximum matching in G using the algorithm of Hopcroft-Karp [6], which runs in time $O\big(|E|\sqrt{|U|}\big) = O(n^{2.5})$. Let $M \subseteq E$ be the resulting maximum matching in G. Define $Q^- \subseteq U^-$ as the set of unmatched vertices. A path in G is *alternating* if the edges of the path alternate between matched and unmatched edges. Let $S \subseteq V^- \cup V^+$ be the set of vertices which are members of alternating paths starting from any vertex in Q^-. Finally, set $D = \{v \in V \mid v^- \in S, v^+ \notin S\}$. We claim D is an anti-chain of maximum size.

Conceptually, suppose that G is a directed network flow graph. Modify G by adding two new vertices s and t and add the directed edges (s, v^-) and (v^+, t), each with capacity one for all $v \in V$. Finally, direct all edges from V^- to V^+ with infinite capacity. By the max-flow

min-cut Theorem, the maximum matching M, induces a minimum s-t cut, which is the cut $(s + S, t + U \setminus S)$, where S is defined above. Indeed, $s + S$ is the reachable set from s in the residual graph for G. To see why D is an anti-chain, suppose that there exist two comparable elements $v, u \in D$. This implies that $v^-, u^- \in S$ and $v^+, u^+ \notin S$. Assume without loss of generality that $v \prec u$. This implies that (v^-, u^+) is an edge of the network flow graph G with infinite capacity that is in the cut $(s+S, V \setminus S+t)$. This contradicts the finiteness of the cut capacity.

We next prove that $|D|$ is maximum. Note that an element $v \in V$ is not in D if $v^- \notin S$ or $v^+ \in S$. If $v^- \notin S$ then (s, v^-) is in the cut. Similarly, if $v^+ \in S$ then (v^+, t) is in the cut. Since the minimum cut has capacity $|M|$, there are at most $|M|$ such vertices, which implies that $|D| \geq n - |M|$.

On the other hand, a chain cover $\mathcal{C}$ for $(V, \prec)$ can be constructed from M. Given $(V, \prec)$, create a DAG H with vertex set V. We add the directed edge (u, v) to H when $u \prec v$.[1] Now an edge (u^-, v^+) in M corresponds to the edge (u, v) of H. As such, a matching corresponds to a collection of edges in H, where every vertex appears at most twice. Since H is a DAG, it follows that M corresponds to a collection of paths in H. The end vertex x of such a path corresponds to a vertex $x^- \in V^-$ that is unmatched (as otherwise, the path can be extended), and this is the only unmatched vertex on this path. There are at most $n - |M|$ unmatched vertices in V^-, which implies that $|\mathcal{C}| \leq n - |M|$. Hence, D is an anti-chain with $|D| \geq n - |M| \geq |\mathcal{C}|$. Additionally, recall that for any anti-chain D', $|D'| \leq |\mathcal{C}|$. These two inequalities imply that D is of maximum possible size. □

Remark 2 *As described above, the edges of the matching M correspond to a collection of edges in the DAG H. These edges together form a collection of vertex-disjoint paths which cover the vertices of H, and this is the minimum possible number of paths needed to cover the vertices.*

2.2 Computing anti-chains on implicit posets

Here, we focus on computing anti-chains in posets, in which comparability of two elements are efficiently computable. Our main observation is that one can use range searching data structures to run the Hopcroft-Karp bipartite matching algorithm faster [6]. This observation goes back to the work of Efrat et al. [4], where they study the problem of computing a perfect matching M in a weighted bipartite graph G such that the maximum weight edge in M is minimized. They focus on solving the decision version of the problem: given a parameter r, is there a perfect matching M with maximum edge weight at most r?

Theorem 3 ([4, Theorem 3.2]) *Let G be a bipartite graph on n vertices with bipartition $A \cup B$, and $r > 0$ a parameter. For any subset $U \subseteq B$ of m vertices, suppose one can construct a data structure $\mathcal{D}(B)$ such that:*

(i) Given a query vertex $v \in A$, $\mathcal{D}(B)$ returns a vertex $u \in B$ such that the wight of the edge (u, v) is at most r (or reports that no such element in B exists) in $T(m)$ time.

(ii) An element of B can be deleted from $\mathcal{D}(B)$ in $T(m)$ time.

(iii) $\mathcal{D}(B)$ can be constructed in $O\big(m \cdot T(m)\big)$ time.

Then one can decide if there is a perfect matching M in G, such that all edges in M have weight at most r, in $O\big(n^{1.5} \cdot T(n)\big)$ time.

Recently, Cabello and Mulzer [1] also use a similar framework as described above for computing minimum cuts in disk graphs in $\widetilde{O}(n^{1.5})$ time. We show that the above framework can also be extended to computing anti-chains, with a small modification to the data structure requirements.

Theorem 4 *Let $(V, \prec)$ be a poset, where $n = |V|$. For any subset $P \subseteq V$ of m elements, suppose one can construct a data structure $\mathcal{D}(P)$ such that:*

(i) Given a query $v \in V$, $\mathcal{D}(P)$ returns an element $u \in P$ with $v \prec u$ (or reports that no such element in P exists) in $T(m)$ time.

(ii) An element can be deleted from $\mathcal{D}(P)$ in $T(m)$ time.

(iii) $\mathcal{D}(P)$ can be constructed in $O\big(m \cdot T(m)\big)$ time.

Then one can compute the maximum size anti-chain for $(V, \prec)$ in $O\big(n^{1.5} \cdot T(n)\big)$ time.

Proof. Create the vertex set $U = V^- \cup V^+$ of the bipartite graph $G = (U, E)$ associated with $(V, \prec)$. The neighborhood of a vertex in the bipartite graph can be found by constructing and querying the data structure $\mathcal{D}$. Recall that in each iteration of the maximum matching algorithm of Hopcroft-Karp [6], a BFS tree is computed in the residual network of G. Such a tree can be computed in $O(nT(n))$, as can be easily verified (the BFS algorithm is essentially described below). Furthermore, the algorithm terminates after $O(\sqrt{n})$ iterations, which implies that one can compute the maximum matching in G in $O(n^{1.5} \cdot T(n))$ time. See Efrat et al. [4] for details. Let M be the matching computed.

By Lemma 1, computing the maximum anti-chain reduces to computing the set of vertices which can be reached by alternating paths originating from unmatched vertices in V^-. Call this set of vertices S, as in Lemma 1.

To compute S, we do a BFS in the residual network of G. To this end, build the data structure $\mathcal{D}(V^+)$. Start

[1] Equivalently, H is the transitive closure of the Hasse diagram for $(V, \prec)$.

at an arbitrary unmatched vertex $v \in V^-$, add it to S, and query $\mathfrak{D}(V^+)$ to travel to a neighbor $u \in V^+$ along an unmatched edge. Add u to S and delete u from $\mathfrak{D}(V^+)$. Travel back to a vertex x in V^- using an edge of the matching M (if possible) and add x to S. This process is iterated until the alternating path has been exhausted. Then, restart the search from v (if v has any remaining unmatched neighbors) or another unmatched vertex of V^-. Observe that each vertex in V^+ is inserted and deleted at most once from the data structure $\mathfrak{D}$. Furthermore, each query to $\mathfrak{D}$ can be charged to a vertex deletion. Hence, S can be computed in $O(n \cdot T(n))$ time.

Given S, in $O(n)$ time we can compute a maximum anti-chain $D = \{v \in V \mid v^- \in S, v^+ \notin S\}$. □

3 Applications

3.1 Largest Pareto-optimal subset of points

Definition 5 *Let P be a set of points in $\mathbb{R}^d$. A point $p \in \mathbb{R}^d$ dominates a point $q \in \mathbb{R}^d$ if $p \geq q$ coordinate wise. The point set P is* **Pareto-optimal** *if no point in P dominates any other point in P.*

Lemma 6 *Let $P \subset \mathbb{R}^d$ be a set of n points. A Pareto-optimal subset of P of maximum size can be computed in $O(n^{1.5}(\log n/\log\log n)^{d-1})$ time.*

Proof. Form the implicit poset $(P, \prec)$ where $q \prec p \iff p$ dominates q. Hence, two elements are incomparable when neither is dominated by the other. As such, computing the largest Pareto-optimal subset is reduced to finding the maximum anti-chain in $(P, \prec)$.

To apply Theorem 4, one needs to exhibit a data structure $\mathfrak{D}$ with the desired properties. For a given query q, finding a point $p \in P$ with $q \prec p$ corresponds to finding a point p which dominates q. Equivalently, such a point in P exists if and only if it lies in the range $[q_1, \infty) \times \ldots \times [q_d, \infty)$. This is a d-sided orthogonal range query. Chan and Tsakalidis's dynamic data structure for orthogonal range searching [2] suffices—their data structure can handle deletions and queries in $T(n) = O((\log n/\log\log n)^{d-1})$ amortized time, and can be constructed in $O(n \cdot T(n))$ time. □

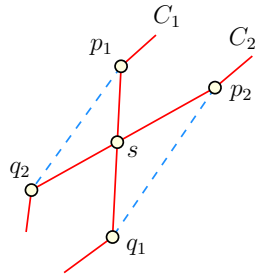
3.1.1 Chain decomposition

Let P be a set of n points in $\mathbb{R}^d$. We would like to decompose P into disjoint chains of dominated points, such that all the points are covered, and the number of chains is minimum. By Remark 2, this can be done by running the algorithm Lemma 6, and converting the bipartite matching to chains. This would take $O(n^{1.5}(\log n/\log\log n)^{d-1})$ time.

As a concrete example, suppose we want to solve a more restricted problem in the planar case—decomposing a given point set into chains of points (that are monotone in both x and y), such that no pair of chains intersect.

Suppose that we have computed a chain decomposition of the points P. Let C_1 and C_2 be two chains of points, each with an edge p_iq_i in C_i (and p_i dominates q_i) for $i = 1, 2$ such that p_1q_1 and p_2q_2 intersect in the plane. An exchange argument shows that by deleting these two edges and adding the edges p_1q_2 to C_1 and p_2q_1 to C_2, we decrease the total length of the chains. Indeed, let s be the intersection point of the edges p_1q_1 and p_2q_2. By the triangle inequality and assuming the points of P are in general position,

$$\begin{aligned}&\|p_1 - q_2\| + \|p_2 - q_1\| \\ &< \|p_1 - s\| + \|s - q_2\| + \|p_2 - s\| + \|s - q_1\| \\ &= \|p_1 - q_1\| + \|p_2 - q_2\|.\end{aligned}$$



As such, suppose we assign a weight to each edge in the bipartite graph G equal to the distance between the two corresponding points. If k is the size of the maximum (unweighted) matching in G, then we can compute a matching of cardinality k with minimum weight by solving a min-cost flow instance on G (using the weights on the edges of G as the costs). This implies that the resulting chain decomposition covers all points of P, and no pair of chains intersect. We obtain the following.

Lemma 7 *Let P be a set of n points in the plane in general position. In polynomial time, one can compute the minimum number of non-intersecting (x, y)-monotone polygonal curves covering the points of P, where every point of P must be a vertex of one of these polygonal curves, and the vertices of the polygonal curves are points of P.*

Remark 8 *If we do not require the collection of polygonal curves to be non-intersecting, there is a much simpler algorithm. Create a directed acyclic graph $H = (P, E)$, where $(p, q) \in E$ if p dominates q. Observe that all of the points $S \subseteq P$ in H with out-degree zero form a polygonal curve. We add this curve to our collection and recursively compute the set of curves on the residual graph $H \setminus S$. While the number of polygonal curves is minimum, the resulting curves may intersect.*

3.2 Largest loose subset of regions

Definition 9 *Let $\mathcal{D}$ be a collection of n regions in $\mathbb{R}^d$. Such a collection $\mathcal{D}$ is* loose *if no region of $\mathcal{D}$ is fully contained inside another region of $\mathcal{D}$.*

Lemma 10 *Let $\mathcal{D}$ be a collection of n regions in $\mathbb{R}^d$. For any subset $R \subseteq \mathcal{D}$ of m regions, suppose one can construct a data structure $\mathfrak{D}(R)$ such that:*

(i) Given a query $\mathsf{d} \in \mathcal{D}$, $\mathfrak{D}(R)$ returns a region $\mathsf{d}' \in R$ with $\mathsf{d}' \subseteq \mathsf{d}$ (or reports that no such region in R exists) in $Q(m)$ time.

(ii) A region can be deleted from $\mathfrak{D}(R)$ in $Q(m)$ time.

(iii) $\mathfrak{D}(R)$ can be constructed in $O\big(m \cdot Q(m)\big)$ time.

Then one can compute the largest loose subset of $\mathcal{D}$ in $O\big(n^{1.5} \cdot Q(n)\big)$ time.

Proof. Form the implicit poset $(\mathcal{D}, \prec)$, where $\mathsf{d}' \prec \mathsf{d}$ $\iff$ the region d is contained in the interior of d'. In particular, a subset of regions are loose if and only if they form an anti-chain in $(\mathcal{D}, \prec)$.

The proof now follows by considering the poset $(\mathcal{D}, \prec)$ described above and applying Theorem 4 using the data structure $\mathfrak{D}$. □

3.2.1 Largest loose subset of disks

We show how to compute the largest loose subset when the regions are disks in the plane. To apply Lemma 10, we need to exhibit the required dynamic data structure $\mathfrak{D}$.

Lemma 11 *Let $\mathcal{D}$ be a set of n disks in the plane. There is a dynamic data structure $\mathfrak{D}$, which given a query disk q, can return a disk $\mathsf{d}' \in \mathcal{D}$ such that $\mathsf{d}' \subseteq \mathsf{q}$ (or report that so such disk exists) in $O(\log^2 n)$ deterministic time. Insertion and deletion of disks cost $O(\log^{10+\varepsilon} n)$ amortized expected time, for all $\varepsilon > 0$.*

Proof. Associate each disk $\mathsf{d} \in \mathcal{D}$, which has center c_d and radius r_d, with a weighted distance function $\delta_\mathsf{d} : \mathbb{R}^2 \to \mathbb{R}$, where $\delta_\mathsf{d}(p) = \|\mathsf{c}_\mathsf{d} - p\| + r_\mathsf{d}$. Observe that a disk d is contained inside the interior of a disk q if and only if $\delta_\mathsf{d}(\mathsf{c}_\mathsf{q}) \leq r_\mathsf{q}$. For a query disk q, our goal will be to compute $\arg\min_{\mathsf{d} \in \mathcal{D}} \delta_\mathsf{d}(\mathsf{c}_\mathsf{q})$. After finding such a disk d', return that $\mathsf{d}' \subseteq \mathsf{q}$ if and only if $\delta_{\mathsf{d}'}(\mathsf{c}_\mathsf{q}) \leq r_\mathsf{q}$.

Hence, the problem is reduced to dynamically maintaining the function $F(p) = \min_{\mathsf{d} \in \mathcal{D}} \delta_\mathsf{d}(p)$, for all $p \in \mathbb{R}^2$, under insertions and deletions of disks. Equivalently, F is also the lower envelope of the xy-monotone surfaces defined by $\{\delta_\mathsf{d} \mid \mathsf{d} \in \mathcal{D}\}$ in $\mathbb{R}^3$. This problem was studied by Kaplan et al. [7]: They prove that if F is defined by a collection of additively weighted Euclidean distance functions, then F can be computed for a given query p in $O(\log^2 n)$ time. Furthermore, updates can be handled in $O\big(2^{O(\alpha(\log n)^2)} \log^{10} n\big)$ time, where $\alpha(n)$ is the inverse Ackermann function. □

Corollary 12 *Let $\mathcal{D}$ be a set of n disks in the plane. The largest loose subset of disks can be computed in $O(n^{1.5} \log^{10+\varepsilon} n)$ expected time, for all $\varepsilon > 0$.*

Proof. Follows from Lemma 10 in conjunction with the data structure described in Lemma 11. □

Remark 13 *By Remark 2, one can decompose a given set of n disks into the minimum number of disjoint* towers*, where each tower is a sequence of disks of the form $d_1 \subseteq d_2 \subseteq \cdots \subseteq d_t$. The resulting running time is as stated in Corollary 12.*

3.3 Largest subset of non-crossing rectangles

Definition 14 *A collection $\mathcal{R}$ of axis-aligned rectangles are non-crossing if the boundaries of every pair of rectangles in $\mathcal{R}$ intersect at most twice.*

Lemma 15 *Let $\mathcal{R}$ be a set of n axis-aligned rectangles in the plane. A non-crossing subset of $\mathcal{R}$ of maximum size can be computed in $O(n^{1.5}(\log n/\log\log n)^3)$ time.*

Proof. For each rectangle $R \in \mathcal{R}$, let $\mathsf{t}(R)$ and $\mathsf{b}(R)$ denote the y-coordinate of the top and bottom sides of R, respectively. Similarly, $\mathsf{l}(R)$ and $\mathsf{r}(R)$ denotes the x-coordinate for the left and right sides of R.

Form the poset $(\mathcal{R}, \prec)$ where

$$R' \prec R \iff [\mathsf{l}(R), \mathsf{r}(R)] \subseteq [\mathsf{l}(R'), \mathsf{r}(R')] \text{ and } [\mathsf{b}(R'), \mathsf{t}(R')] \subseteq [\mathsf{b}(R), \mathsf{t}(R)].$$

Observe two rectangles R and R' are incomparable if and only if the boundaries of R and R' intersect at most twice. In particular, the largest subset of non-crossing rectangles corresponds to the largest anti-chain in $(\mathcal{R}, \prec)$.

To apply Theorem 4, we need a dynamic data structure which, given a rectangle $R \in \mathcal{R}$, returns any rectangle in $R' \in \mathcal{R}$ where $R \prec R'$. Equivalently, we want to return a rectangle $R' \in \mathcal{R}$ such that $[\mathsf{l}(R'), \mathsf{r}(R')] \subseteq [\mathsf{l}(R), \mathsf{r}(R)]$ and $[\mathsf{b}(R), \mathsf{t}(R)] \subseteq [\mathsf{b}(R'), \mathsf{t}(R')]$. To do so, map each rectangle $R \in \mathcal{R}$ to the point $(\mathsf{l}(R), \mathsf{r}(R), \mathsf{t}(R), \mathsf{b}(R)) \in \mathbb{R}^4$. The query of interest reduces to a 4-sided orthogonal range query in $\mathbb{R}^4$. Chan and Tsakalidis's dynamic data structure for orthogonal range searching [2] supports such queries and updates in time $O((\log n/\log\log n)^3)$, implying the result. □

3.4 Largest subset of isolated points

Let L be a set of n lines in the plane. We assume that no line in L is vertical and L may not necessarily be in general position. Let P be a set of n points lying on the lines of L.

Definition 16 *Given a set of lines L and points P lying on L, a $p \in P$ can reach a point $q \in P$ if it possible for p to reach q by traveling from left to right along lines in L. The set P is isolated if no point in P can reach another point in P.*

The partial ordering. Fix the collection of lines L. Given P, create the poset $(P, \prec)$, where $p \prec q \iff p$ can reach q using the lines of L. Observe that any subset of isolated points directly corresponds to an anti-chain in $(P, \prec)$.

Lemma 17 *Let P be a collection of n points in the plane lying on a set L of n lines. The largest subset of isolated points can be computed in $O(n^3)$ time.*

Proof. We can assume that every point of P lies on at least two lines of L. If not, shift such a point p to the right along the line it lies on, until p encounters an intersection.

Start by computing the arrangement $\mathcal{A}(L)$ of the lines L. Next, construct a directed graph G with vertex set equal to the vertices of $\mathcal{A}(L)$. By assumption, P is a subset of the vertices of G. The edges of G consist of the edges of the arrangement $\mathcal{A}(L)$ (any edges of $\mathcal{A}(L)$ which are half-lines are ignored). For each edge of $\mathcal{A}(L)$ with endpoints u, v, we direct the edge in G from u to v when u has a smaller x-coordinate than v. Next, for each $p \in P$, determine the set of points of P reachable from p by performing a BFS in G. Thus, given any two points, we can determine if they are comparable in $O(1)$ time. Apply Lemma 1 to obtain the largest isolated subset.

To analyze the running time, note that computing the arrangement $\mathcal{A}(L)$ and constructing G can be done in $O(n^2)$ time. A BFS from n points in G costs $O(n^3)$ time total. Finally, the largest isolated subset can be found in $O(n^{2.5})$ time by Lemma 1. □

References

[1] S. Cabello and W. Mulzer. Minimum cuts in geometric intersection graphs. *CoRR*, abs/2005.00858, 2020.

[2] T. M. Chan and K. Tsakalidis. Dynamic orthogonal range searching on the ram, revisited. In *33rd Symp. on Comput. Geom.* (SoCG), pages 28:1–28:13, 2017.

[3] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[4] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[5] S. Felsner and L. Wernisch. Maximum k-chains in planar point sets: Combinatorial structure and algorithms. *SIAM J. Comput.*, 28(1):192–209, 1998.

[6] 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, 1973.

[7] H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. In P. N. Klein, editor, *28th Symp. on Discrete Algorithms* (SODA), pages 2495–2504. SIAM, 2017.

[8] J. Matoušek and E. Welzl. Good splitters for counting points in triangles. *J. Algorithms*, 13(2):307–319, 1992.

[9] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[10] M. Segal and K. Kedem. Geometric applications of posets. *Comput. Geom.*, 11(3-4):143–156, 1998.

Discrete Helly type theorems

Frederik Brinck Jensen* Aadi Joshi† Saurabh Ray‡

Abstract

We show the following discrete Helly-type results in the plane. Let P be a set of points and $\mathcal{D}$ a family of convex pseudodisks in the plane s.t. every triple of pseudodisks in $\mathcal{D}$ intersects at a point in P. Then, two of the points in P *hit* all pseudodisks in $\mathcal{D}$. Three similar results that we prove with the roles of points and regions exchanged are the following: let P be a set of points and $\mathcal{H}$ a set of halfspaces in the plane s.t. every triple of points from P is covered by some halfspace in $\mathcal{H}$. Then, two of the halfspaces in $\mathcal{H}$ cover all points in P. If, instead, every pair of points in P is covered by some halfspace in $\mathcal{H}$, we show that three of halfspaces in $\mathcal{H}$ suffice to cover all points in P. Finally, we show that if every pair of points is covered by an axis-parallel rectangle, then five rectangles are required to cover all points.

1 Introduction

Let $\mathcal{C}$ be a finite collection of convex sets in $\mathbb{R}^d$. Helly's theorem states that if every $d+1$ of these sets intersect at a common point in $\mathbb{R}^d$ then all the convex sets in $\mathcal{C}$ intersect at a common point in $\mathbb{R}^d$.

It is natural to ask if a discrete version of Helly's theorem is true. Instead of requiring that every $d+1$ of convex sets intersect at some point in $\mathbb{R}^d$, suppose that we require that they intersect at some point in a discrete set of points P. Then, can we conclude that all the convex sets intersect at some point in P? Unfortunately, this statement is not true even if we require that every k of the convex sets intersect at a point in P for some large constant $k \geq d+1$ and we want to conclude that all convex sets in $\mathcal{C}$ can be hit by some large constant number of points. To see this, consider a set P of n points in convex position in $\mathbb{R}^d$ and let $\mathcal{C}$ be the set of the convex hulls of every subset of P of size greater than $n - \frac{n}{k}$. Then the total size (in terms of the number of points of P contained) of any subset of k sets in $\mathcal{C}$ is more than $(k-1)\,n$ and thus they must have a common point in P. On the other hand, no subset of P of size less than n/k can hit all the convex sets in $\mathcal{C}$.

We show that such a statement is true for some simple regions in the plane. We believe that the phenomenon is more general but we currently lack tools for proving such results.

*New York University Abu Dhabi, fj414@nyu.edu
†New York University Abu Dhabi, aj1566@nyu.edu
‡New York University Abu Dhabi, sr194@nyu.edu

2 Discrete Helly type theorem for halfplanes

We start with a simple Helly-type result for halfspaces in the plane.

Theorem 1 *Let P be a finite set of n points and $\mathcal{H}$ a set of halfspaces in the plane. If every subset of 3 points in P belongs to some halfspace $H \in \mathcal{H}$ then there exists two halfspaces $H_1, H_2 \in \mathcal{H}$ whose union covers P.*

Proof. Let $P_{CH} \subseteq P$ denote the subset of points in P that lie on the convex hull $\mathrm{CH}(P)$ of P. Consider the halfspace $H_1 \in \mathcal{H}$ containing the largest number of points from P_{CH}. Note that since there is a halfspace in $\mathcal{H}$ covering any triple of points in P_{CH}, H_1 contains at least three points of P_{CH}.

If H_1 contains all points in P_{CH}, then H_1 contains all points in P and the theorem follows. Otherwise, the boundary of H_1 intersects two edges of the boundary of $\mathrm{CH}(P)$. Let p and q be endpoints of these two edges contained in H_1. See figure 1. Note that p and q cannot be the same point since this would mean that H_1 contains only one point of P_{CH} and we argued earlier that H_1 contains at least three points of P_{CH}.

The line through p and q splits $\mathrm{CH}(P)$ into two regions, one of which is covered by H_1. Let A be the region covered by H_1 and let B be the other region. Let $r \in P_{CH}$ be a point not contained in H_1. By assumption, there exists a halfspace, H_2, that contains p, q, and r.

Since H_2 contains p and q, H_2 covers either A or B. If H_2 covered A, then it would contain all points of P_{CH} in H_1 and the additional point $r \in P_{CH}$. This would contradict the maximality of H_1. Thus H_2 must cover B. Thus $H_1 \cup H_2$ covers $\mathrm{CH}(P)$ and the theorem follows. □

We now show that the '3' in Theorem 1 is tight i.e., it cannot be replaced by '2'. To this end, we construct a set of points P and a set of halfspaces $\mathcal{H}$ so that every pair of points in P is covered by a halfspace in $\mathcal{H}$ and yet no pair of halfspaces in $\mathcal{H}$ cover all points in P.

Figure 2 shows a disk D and three arcs with a very large radius of curvature so that any tangent to any of the arcs passes through the disc D and does not intersect any of the other arcs. We construct a point set P with

n points by distributing $n/3$ points uniformly on each of the three arcs. We define the set of halfspaces $\mathcal{H}$ as follows. For each point p on some arc l_i, $i \in \{0, 1, 2\}$, let H_p be a halfspace not containing p and containing all other points on l_i so that the boundary of H_p is parallel and arbitrarily close to the tangent to l_i at p. Note that H_p contains all points on l_j where $j = (i+1) \mod 3$ and does not contain any of the points in l_k where $k = (i-1) \mod 3$. $\mathcal{H}$ is the set $\{H_p : p \in P\}$.

For any two points $p, q \in P$, we argue that there is a halfspace in $\mathcal{H}$ containing both p and q. There are two cases depending on whether p and q lie on the same arc or on different arcs. If p and q lie on the same arc l_i, then H_r where r is any point on l_k where $k = (i-1) \mod 3$ contains both p and q. If p and q lie on different arcs then without loss of generality, assume that p lies on l_i and q lies on l_j where $j = (i+1) \mod 3$. Then H_r, where r is any point on l_i other than p, contains both p and q.

Now, we argue that no two halfspaces in $\mathcal{H}$ cover all points in P. To see this consider any pair of halfspaces H_p and H_q. If both p and q lie on the same arc l_i then none of the points in l_k where $k = (i-1) \mod 3$ are covered by $H_p \cup H_q$. If p and q lie on different arcs, we assume without loss of generality that p lies on an arc l_i and q lies on l_j where $j = (i+1) \mod 3$. Then $H_p \cup H_q$ does not cover p.

Remark. The above example also improves a result from [1]. Lemma 17 of [1] shows that given a set P of n points in the plane, it is not always possible to hit all halfspaces in the plane containing more than ϵn points of P with just two points in P if $\epsilon < 3/5$. Our construction improves the bound $3/5$ to $2/3$ since all the halfspaces in our example contain $2n/3 - 1$ points and it is easily seen that no two points hit all halfspaces.

In our example, two halfspaces barely fail to cover all the points when every pair of points in P is covered by a halfspace. It seems intuitive that three halfspaces should suffice to cover all points under this constraint. Indeed this is true and we show this later in Theorem 7.

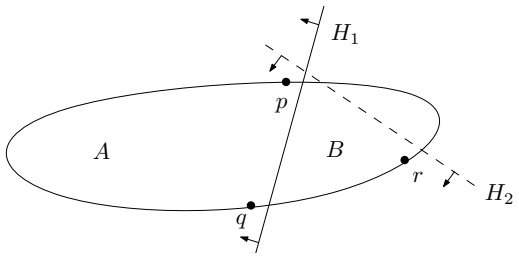


Figure 1: Let H_1 be the halfspace that contains the most points in P_{CH}. As depicted, the only positioning of any halfspace H_2 containing p, and q, and a point $r \notin H_1$ that avoids parts of $P \setminus H_1$ must contain more points on the convex hull of P than H_1, which is a contradiction. The dashed line, H_2, can therefore not exist and we are left with a halfspace that covers the remaining points not contained in H_1.

3 Discrete Helly type theorem for convex pseudodisks

A set of simply connected regions in the plane form a family of pseudodisks if the boundaries of any pair of regions either do not intersect or intersect at exactly two points. Furthermore, there are no tangential intersections, i.e., at each intersection the boundaries properly cross. Examples of families of pseudodisks include disks, squares, unit height rectangles, and homothets of a convex region.

Let $P \subset \mathbb{R}^2$ be a set of points and $\mathcal{D}$ a family of convex pseudodisks in the plane. We will show that if every three pseudodisks in $\mathcal{D}$ intersect at a point in P, then there exist two points in P that together hit all the pseudodisks in $\mathcal{D}$.

We first need a definition and some lemmas.

Definition 1 *For any disk $D \in \mathcal{D}$ we define its core* $\text{core}(D)$ *with respect to P as the convex hull of $D \cap P$. See Figure 3.*

Even though the core of a pseudodisk D is defined with respect to the point set P, we will skip the reference to P when it is clear from the context.

Note that any pseudodisk $D \in \mathcal{D}$ contains its core since D is convex. Let $\mathcal{C}$ be the set of cores of the disks $D \in \mathcal{D}$.

Lemma 2 *The intersection of all cores in $\mathcal{C}$ is non-empty.*

Proof. Since every triple of convex pseudodisks in $\mathcal{D}$ intersects at some point $p \in P$ that belongs to the cores

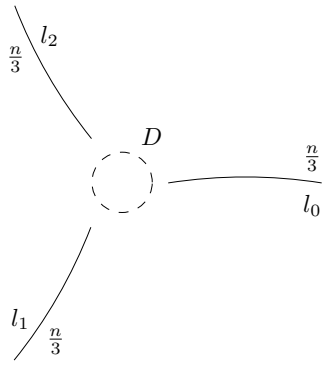


Figure 2: Counterexample that shows the tightness of Theorem 1.

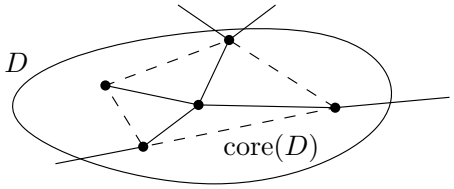


Figure 3: The dashed line outlines core(D).

of all three of them, all triples of cores in $\mathcal{C}$ have a non-empty intersection. Thus, by Helly's theorem all cores in $\mathcal{C}$ intersect at a point in $\mathbb{R}^2$ (which may not be a point in P). □

The next lemma follows from Lemma 5 in [2] by using an empty set as the set of *compulsory edges*.

Lemma 3 *There exists a straight-edge plane triangulation on P denoted $\mathcal{T}$ such that the points and edges inside any pseudodisk in $\mathcal{D}$ form a connected subgraph of $\mathcal{T}$.*

Lemma 5 in [2] is for arbitrary non-convex k-admissible regions (which includes pseudodisks) and therefore allows the edges to be curved. However, for convex pseudodisks, it does yield a straight-edge drawing. In fact, it shows that any maximal subset of the $\binom{n}{2}$ edges defined by P which are pairwise non-crossing and which do not cut across any of the cores of the regions form such a triangulation. Note that if $\mathcal{D}$ is a set of circular disks in the plane, then the Delaunay triangulation of P provides the triangulation claimed in the lemma above.

Lemma 4 *If the core of some $D \in \mathcal{D}$ intersects an edge $e \in \mathcal{T}$, then D must contain at least one of the endpoints of e.*

Proof. Since the edges of $\mathcal{T}$ are straight line segments, the points and edges inside any core in $\mathcal{C}$ also form a connected sub-graph of $\mathcal{T}$. If the core of a disk $D \in \mathcal{D}$ intersects an edge $e \in \mathcal{T}$ but does not contain either endpoint of e, then we obtain a contradiction since the edges lying in core(D) cannot form a connected subgraph of $\mathcal{T}$. Thus core(D) must contain an endpoint of e. □

Lemma 5 *There exist two points p, $q \in P$ that hit all disks in $\mathcal{D}$.*

Proof. Let $x \in \mathbb{R}^2$ be a point in the common intersection $\cap_{D\in\mathcal{D}}$ core(D) of all cores. By lemma 2, such a point x exists. Let T be the triangle in the triangulation $\mathcal{T}$ containing x. Since all the cores contain x, all cores intersect the edges of T and thus, by Lemma 4 contain at least one of the three corners of T. In other words, the three corners of T hit all pseudodisks.

We now show that one of the corners is redundant and can be dropped. Assume, for contradiction that all three corners of T are necessary i.e. for each corner there exists a pseudodisk that is hit only by that corner among the three corners. These three pseudodisks intersect inside T at x and, by definition, at some point in $p \in P$ which must lie outside T. Since the intersection of the three convex pseudodisks is convex, all three disk contain the segment joining x and p. Therefore, they all intersect some edge $e \in T$. However, by Lemma 4 this means that all three disks are hit by the two endpoints of e contradicting the assumption that all three corners are necessary for hitting the three disks.

□

Thus we have proved the following theorem.

Theorem 6 *Given a set of points P and set of convex pseudodisks $\mathcal{D}$ in the plane s.t. every triple of pseudodisks in $\mathcal{D}$ intersects at a point in P, there exists a set of two points $\{p, q\} \subseteq P$ which intersects each $D \in \mathcal{D}$.*

Remark. The above theorem implies that given a set of n points and a set $\mathcal{D}$ of convex pseudodisks in the plane, all pseudodisks in $\mathcal{D}$ containing more than $2n/3$ points can be hit by two points. This is because any three sets containing more than $2n/3$ elements of a set of size n must have a common element. This generalizes Theorem 18 of [1] which proves this for disks.

Note that Theorem 6 is true for halfspaces in the plane too since we can think of halfspaces as disks of infinite radius. This can be used to prove the following theorem.

Theorem 7 *Let $\mathcal{H}$ be a set of halfspaces and let P be a set of points in the plane such that for each pair of points in P, there is a halfspace in $\mathcal{H}$ covering both points. Then, three of the halfspaces in $\mathcal{H}$ cover all points in P.*

Proof. Let $\mathcal{H}'$ be a set of halfspaces consisting of the complements of the halfspaces in $\mathcal{H}$. Assume for contradiction that no three halfspaces in $\mathcal{H}$ cover all points in P. This means that every triple of halfspaces in $\mathcal{H}'$ intersects at a point in P. Since the halfspaces in $\mathcal{H}'$ can be thought of as disks of infinite radius, we can apply Theorem 6 to $\mathcal{H}'$ and P to conclude that two points $p, q \in P$ hit all halfspaces in $\mathcal{H}'$. This implies that each halfspace in $\mathcal{H}$ avoids at least one of the points in $\{p, q\}$. This contradicts the assumption that for every pair of points in P, there is some halfspace in $\mathcal{H}$ that contains both. □

4 Discrete Helly type theorem for disks

Theorem 8 *Let D be a set of disks and let P be a set of points in the plane such that every tripe of points in P is covered by a disk in D. Then, two disks in D cover all points in P.*

Proof. Let D_s be the disk of smallest radius that covers all points in P. Note that D_s may not be in D. D_s must contain two or three points on its boundary. Let D_1 be the disk in D that covers all points on the boundary of D_s and covers the greatest number of points in P out of all such disks. Then, D_1 either covers all points in P, in which case we are done, or there is a region in D_s that is not covered by D_1. This region lies on one side of the chord defined by two points p_1, $p \in P$ on the boundary of D_s. □

5 Open Problems

It is natural to ask whether analogues of Theorems 1 and 6 are true when halfspaces and pseudodisks are replaced by other regions. We mention two that we find intriguing.

Open problem 1**.** Let $\mathcal{D}$ be a set disks and let P be a set of points in $\mathbb{R}^2$ s.t. for every triple of points in P, there is a disk in $\mathcal{D}$ covering the three points. How many disks of $\mathcal{D}$ suffice to cover all points (in the worst case)? We believe that two disks in $\mathcal{D}$ suffice.

Open Problem 2**.** Let $\mathcal{R}$ be a set of axis-parallel rectangles and let P be a set of points in the plane s.t. every triple of rectangles in $\mathcal{R}$ intersects at a point in P. How many points from P suffice to hit all rectangles in $\mathcal{R}$.

References

[1] Pradeesha Ashok, Umair Azmi, and Sathish Govindarajan. Small strong epsilon nets. *Comput. Geom.*, 47(9):899–909, 2014.

[2] Evangelia Pyrga and Saurabh Ray. New existence proofs epsilon-nets. In Monique Teillaud, editor, *Proceedings of the 24th ACM Symposium on Computational Geometry, College Park, MD, USA, June 9-11, 2008*, pages 199–207. ACM, 2008.

If You Must Choose Among Your Children, Pick the Right One

Brittany Terese Fasy[†*] Benjamin Holmgren[†] Bradley McCoy[†] David L. Millman[†]

Abstract

Given a simplicial complex K and an injective function f from the vertices of K to $\mathbb{R}$, we consider algorithms that extend f to a discrete Morse function on K. We show that an algorithm of King, Knudson and Mramor can be described on the directed Hasse diagram of K. Our description has a faster runtime for high dimensional data with no increase in space.

1 Introduction

Milnor's classical Morse theory provides tools for investigating the topology of smooth manifolds [16]. In [9], Forman showed that many of the tools for continuous functions can be applied in the discrete setting. Inferences about the topology of a CW complex can be made from the number of critical cells in a Morse function on the complex.

Given a Morse function one can interpret the function in many ways. Switching interpretations is often revealing. In this paper, we think of a discrete Morse function in three different ways. Algebraically, a Morse function is a function from the faces of a complex to the real numbers, subject to certain inequalities. Topologically, a Morse function is a pairing of the faces such that the removal of any pair does not change the topology of the complex. Combinatorially, a Morse function is an acyclic matching in the Hasse diagram of the complex, where unmatched faces correspond to critical cells.

Discrete Morse theory can be combined with persistent homology to analyze data, see [1, 2, 4, 5, 7, 8, 13]. When dealing with data, we have the additional constraint that vertices have function values assigned. For complexes without any preassigned function values, Joswig and Pfetsch showed that finding a Morse function with a minimum number of critical cells is NP-Hard [12]. Algorithms that find Morse functions with relatively few critical cells have been explored in [11,15,17].

In this work, we consider the algorithm Extract, Algorithm 1, given in [13]. Extract takes as input a simplicial complex and an injective function from the vertices to the reals, and returns a discrete Morse function, giving topological information about the complex. We show that a subalgorithm of Extract, ExtractRaw can be simplified by considering the directed Hasse diagram. This simplification leads to an improved runtime and no change in space. The paper is organized as follows, in Section 2, we provide the definitions that will be used in the paper. In Section 3, we describe Extract and analyze the runtime, then, in Section 4, we give our reformulation and show that the runtime is improved from $\Omega(n^2 \log n)$ to $O(dn)$ where n is the number of cells and d is the dimension of K.

[*]Department of Mathematical Sciences, Montana State U.
[†]School of Computing, Montana State U.
{brittany.fasy, bradleymccoy, david.millman}@montana.edu
benjamin.holmgren@student.montana.edu

2 Background

In this section, we provide definitions, notation, and primitive operations used throughout the paper. For a general overview of discrete Morse theory see [14, 19], note that both texts provide a description of Extract originally given in [13]. Extract is the starting point for this work.

In what follows, we adapt the notation of Edelsbrunner and Harer [6] to the definitions of Forman [10]. Here, we work with simplicial complexes, but the results hold for CW complexes. Let K be a simplicial complex with n simplices. For $i \in \mathbb{N}$, denote the i-simplices of K as K_i, the number of simplices in K_i as n_i, and the dimension of the highest dimensional simplex of K as $\dim(K)$.

Let $\sigma \in K$, denote the dimension of σ as $\dim(\sigma)$. Let $p = \dim(\sigma)$ and $\{v_0, v_1, \cdots, v_p\} \subseteq K_0$ be the zero-simplices of σ, then we say $\sigma = [v_0, v_1, \cdots, v_p]$. If $\tau \in K$ is disjoint from σ, then we can define the *join* of σ and τ to be the $(\dim(\sigma) + \dim(\tau) + 1)$-simplex that consists of the union of the vertices in σ and τ, denoted $\sigma * \tau$. We write $\sigma \prec \tau$ if σ is a proper face of τ.

Let $p \in \mathbb{N}$ and consider simplices $\sigma_u, \sigma_v \in K_p$, with $\sigma_u = [u_0, u_1, \ldots, u_p]$ and $\sigma_v = [v_0, v_1, \ldots, v_p]$. Let $f_0 : K_0 \to \mathbb{R}$ be an injective function. Without loss of generality, assume that the zero-simplices of σ_u and σ_v are sorted by function value, that is, we have $f_0(v_i) < f_0(v_j)$ when $0 \le i < j \le p$, similarly for σ_u. We say that σ_u is lexicographically smaller than σ_v, denoted $\sigma_u <_{lex} \sigma_v$, if the vector $\langle f_0(u_p), f_0(u_{p-1}), \ldots, f_0(u_0)\rangle$ is lexicographically smaller than $\langle f_0(v_p), f_0(v_{p-1}), \ldots, f_0(v_0)\rangle$.

The *star* of v in K, denoted $\mathrm{star}_K(v)$, is the set of all simplices of K containing v. The *closed star* of v in K, denoted $\overline{\mathrm{star}}_K(v)$, is the closure of $\mathrm{star}_K(v)$. The *link* of v in K, is denoted as $\mathrm{link}_K(v) := \overline{\mathrm{star}}_K(v) \setminus \mathrm{star}_K(v)$. We define the *lower link* of v, denoted $\mathrm{lowerlink}_K(v)$,

to be the maximal subcomplex of $\text{link}_K(v)$ whose zero-simplices have function value less than $f_0(v)$; the lower link can be computed in $O(n)$ time.

We provide the definition of a Morse function, modified from Forman [10].

Definition 1 (Morse Function) *A function $f : K \to \mathbb{R}$ is a* discrete Morse function*, if for every $\sigma \in K$, the following two conditions hold:*

1. $|\{\beta \succ \sigma | f(\beta) \leq f(\sigma)\}| \leq 1$,
2. $|\{\gamma \prec \sigma | f(\gamma) \geq f(\sigma)\}| \leq 1$.

An intuitive definition is given in [19], "the function generally increases as you increase the dimension of the simplices. But we allow at most one exception per simplex." Let $f : K \to \mathbb{R}$ be a discrete Morse function. A simplex $\sigma \in K$ is *critical* if the following two conditions hold:

1. $|\{\beta \succ \sigma | f(\beta) \leq f(\sigma)\}| = 0$,
2. $|\{\gamma \prec \sigma | f(\gamma) \geq f(\sigma)\}| = 0$.

Simplices that are not critical are called *regular*.

Given a discrete Morse function f on a simplicial complex K, we define the induced *gradient vector field*, or GVF, for short, as $\{(\sigma, \tau) : \sigma \prec \tau, f(\sigma) \geq f(\tau)\}$. Note that σ is a codimension one face of τ. We can gain some intuition for this definition by drawing arrows on the simplicial complex as follows. If σ is regular, a codimension one face of τ, and $f(\tau) \leq f(\sigma)$, then we draw an arrow from σ to τ. Constructing a GVF for a simplicial complex is as powerful as having a discrete Morse function, and is the goal of both Extract and our proposed Algorithm 2.

Next, we define two functions that are helpful when constructing a GVF. The *rightmost face* of σ, denoted $\rho(\sigma)$, is the face of σ with maximum lexicographic value. The *leftmost coface* of σ, denoted $\ell(\sigma)$, is the dimension one coface of σ with minimum lexicographic value. We say σ is a *left-right parent* and we call $\rho(\sigma)$ a *left-right child* if $\ell \circ \rho(\sigma) = \sigma$.

In [10], Forman showed that each simplex in K is exclusively a tail, head, or unmatched. Moreover, the unmatched simplices are critical. Thus, we can partition the simplices of K into *heads* H, *tails* T, and *critical simplices* C, and encode the GVF as a bijection $m : T \to H$. That is, we can represent the GVF for f as the unique tuple (H, T, C, m). We will use this representation throughout our algorithms.

Note that a GVF is a particularly useful construction. It provides a way to reduce the size of a simplicial complex without changing the topology (by cancelling matched pairs), which is constructive for preprocessing large simplicial complexes. See [4, 17] for examples.

We define a consistent GVF as follows:

Definition 2 (Consistent GVF) *Let K be a simplicial complex, and let $f_0 \colon K_0 \to \mathbb{R}$ be injective. Then, we say that a gradient vector field (H, T, C, m) is* consistent *with f_0 if, for all $\varepsilon > 0$, there exists a discrete Morse function $f \colon K \to \mathbb{R}$ such that*

(a) (H, T, C, m) is the GVF corresponding to f.

(b) $f|_{K_0} = f_0$.

(c) $|f(\sigma) - \max_{v \in \sigma} f_0(v)| \leq \varepsilon$.

Let (H, T, C, m) be a GVF. Then, for $r, p \in \mathbb{N}$, a *gradient path*[1] is a sequence of simplices in K:

$$\Gamma = \{\sigma_{-1}, \tau_0, \sigma_0, \tau_1, \sigma_1, \ldots, \tau_r, \sigma_r, \tau_{r+1}\}$$

beginning and ending with critical simplices $\sigma_{-1} \in K_{p+1}$ and $\tau_{r+1} \in K_p$ such that for $0 \leq i < r$, $\tau_i \in K_p$, $\sigma_i \in K_{p+1}$, $m(\tau_i) = \sigma_i$, and $\tau_i \succ \sigma_{i+1} \neq \sigma_i$. We call a path *nontrivial* if $r > 0$.

3 A Discrete Morse Extension of $f_0 \colon K_0 \to \mathbb{R}$

In this section, we give a description of Algorithm 1 (Extract), originally from [13]. This algorithm takes a simplicial complex K, an injective function $f_0 \colon K_0 \to \mathbb{R}$, and a threshold that ignores pairings with small persistence $p \geq 0$; and returns a GVF on K that is consistent with f_0.

Algorithm 1 [13] Extract

Input: A simplicial complex K, injective function $f_0 \colon K_0 \to \mathbb{R}$, and $p \geq 0$
Output: a GVF consistent with f_0

```
1: γ ← ExtractRaw(K, f_0)                          ▷ Algorithm 3
2: for j = 1, 2, ..., dim(K) do
3: |   γ ← ExtractCancel(K, h, p, j, γ)             ▷ Alg. 4
4: return γ
```

Extract uses two subroutines: First, in Line 1 of Algorithm 1 ExtractRaw (given in Algorithm 3) is used to generate an initial GVF on K consistent with f_0. Let (H_0, T_0, C_0, r_0) be this initial GVF. Then, for each dimension ($j = 1$ through $\dim(K)$), the algorithm makes a call to ExtractCancel (given in Algorithm 4) that augments an existing gradient path to remove simplices from C_0 in pairs. For more details, see Appendix A.1.

In the next section, we provide a simpler and faster algorithm to replace ExtractRaw, which dominates the runtime of Extract when $p = 0$ (and in practice, when p is very small). We conclude this section with properties of the output from ExtractRaw:

[1] There is a slight discrepancy between the definition of Forman [10] and KKM [13]. In particular, Forman's definition states the head and the tail of the path are simplices of the same dimension. On the other hand, KKM's usage in the ExtractCancel algorithm expects that the head and tail are different dimensions. Here, we state the definition implied by the usage in KKM.

Theorem 3 (Properies of ExtractRaw) *Let K be a simplicial complex, let $f_0 \colon K_0 \to \mathbb{R}$ be an injective function, and suppose (H, T, C, m) is the output of* ExtractRaw(K, f_0). *Let $\varepsilon > 0$. Then, there exists a discrete Morse function $f \colon K \to \mathbb{R}$ such that the following hold:*

(i) (H, T, C, m) is a GVF consistent with f_0.

(ii) Let $\sigma \in K$. Then, $\sigma \in H$ if and only if σ is a left-right parent.

(iii) For all $\sigma \in H$, $m(\rho(\sigma)) = \sigma$.

(iv) The runtime of ExtractRaw *is $\Omega(n^2 \log n)$.*

4 A Faster Algorithm for ExtractRaw

The main contribution of this paper is ExtractRightChild, which we show is a simplified version of ExtractRaw that has the same output with an improved runtime. This section provides a description of the algorithm, and a proof of the equivalence with ExtractRaw.

4.1 Hasse Diagram Data Structure

We assume that KKM [13] represent K in a *standard Hasse diagram data structure* $\mathcal{H}$, which can be encoded as an adjacency list representation for a graph. Each simplex $\sigma \in K$ is represented by a node in $\mathcal{H}$. We abuse notation and write $\sigma \in \mathcal{H}$ as the corresponding node. Two simplices $\sigma, \tau \in \mathcal{H}$ are connected by an edge from σ to τ if σ is a codimension one face of τ. For a node $\sigma \in \mathcal{H}$, we partition its edges into two sets, $up(\sigma)$ and $down(\sigma)$ as the edges in which σ is a face or coface, respectively.

For $p \in \mathbb{N}$, we denote the nodes of $\mathcal{H}$ corresponding to the p-simplices of K as $\mathcal{H}_p$ and we store each $\mathcal{H}_p$ in its own set that can be accessed in $O(1)$ time. Note that there is no requirement about the ordering of the edges or the nodes in each $\mathcal{H}_p$. See Figure 1 for an example of the data structure.

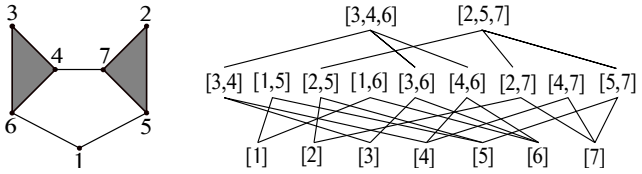


Figure 1: *Left*: A simplicial complex with function values assigned to the vertices. *Right*: The Hasse diagram of the simplicial complex.

For our algorithm, we decorate each node of $\mathcal{H}$ with additional data. For clarity, we denote the decorated data data structure as $\mathcal{H}^*$. Next, we describe the additional data stored in each node and how to initialize the data. Consider $\sigma \in \mathcal{H}_p^*$ and define $\tilde{f}(\sigma) := \max_{v \in \sigma} f_0(v)$. Each node stores $\tilde{f}(\sigma)$, the rightmost child $\rho(\sigma)$ and leftmost parent $\ell(\sigma)$.

Next, we describe how to initialize the data and summarize with the following lemma.

Lemma 4 (Hasse decoration) *Given a simplicial complex K with n simplices and $\dim(K) = d$. The decorated Hasse diagram uses $O(n)$ additional space. We can decorate the Hasse digram K in $O(dn)$ time.*

Proof. We begin by analyzing the space complexity. For each node, we store a constant amount of additional data. Thus, the decorated Hasse diagram uses $O(n)$ additional space.

Next, we analyze the time complexity. To decorate $\mathcal{H}$ for each node $\sigma \in \mathcal{H}$, we must compute $\tilde{f}(\sigma)$, $\rho(\sigma)$, and $\ell(\sigma)$. Let $p = \dim(\sigma)$. We proceed in three steps.

First we compute $\tilde{f}$. In general, computing $\tilde{f}(\sigma)$ takes $O(p)$ time, since there may be no more than p vertices which compose any σ. Let τ_1 and τ_2 be distinct codimension one faces of σ. Observe that $\tilde{f}(\sigma) = \max(\tilde{f}(\tau_1), \tilde{f}(\tau_2))$. Thus, if we know the function values for $\mathcal{H}_{p-1}$, we can compute and store all function values of all nodes in $\mathcal{H}$ in $\Theta(n)$ time.

Second we compute $\rho(\sigma)$ by brute force. We iterate over all edges in $down(\sigma)$ to find its largest face under lexicographic ordering. Since a p-simplex σ has $p + 1$ down edges, computing $\rho(\sigma)$ for σ takes $O(p)$ time. As $\dim(K) = d$, and there are n nodes, we can then compute ρ for all nodes in $O(dn)$ time.

Third, we compute ℓ, also by brute force. We iterate over all edges in $up(\sigma)$ to find its smallest lexicographical coface. While we cannot bound $up(\sigma)$ as easily as $down(\sigma)$, we do know that when computing ℓ we can charge each edge in the Hasse diagram for one comparison. Observe that when computing ρ, we can similarly charge each comparison to an edge. Then, from computing ρ, we know the total number of comparisons is $O(dn)$. Thus, the total number of comparisons for computing ℓ is also $O(dn)$.

As each step takes $O(dn)$ time, decorating $\mathcal{H}$ takes $O(dn)$ time. □

4.2 Algorithm Description

Next, we describe the main algorithm. Given a simplicial complex K (represented as a Hasse diagram), and an injective function $f_0 : K_0 \to \mathbb{R}$, ExtractRightChild computes a GVF consistent with f_0.

Algorithm 2 has three main steps. First, we create a decorated Hasse diagram. Second, we process each level of the Hasse diagram from top to bottom. For

Algorithm 2 ExtractRightChild

Input: simp. complx. K, injective fcn. $f_0 : K_0 \to \mathbb{R}$
Output: a GVF consistent with f_0

```
1:  H* ← decorate the Hasse diagram of K  ▷ Lem. 4
2:  T ← ∅, H ← ∅, C ← ∅, m ← ∅
3:  for i = dim(K) to 1 do
4:      for σ ∈ H*_i do
5:          if σ is assigned then
6:              continue
7:          if σ is a left-right parent then
8:              Add ρ(σ) to T; Add σ to H
9:              Add (ρ(σ), σ) to m
10:             Mark σ and ρ(σ) as assigned
11:         else
12:             Add σ to C
13:             Mark σ as assigned
14: Add any unassigned zero-simplices to C
15: return (T, H, C, r)
```

each unassigned simplex, we check for a left-right parent node, and use the results to build up a GVF. Third, we process unassigned zero-simplices. See Figure 2 for an example.

4.3 Analysis of ExtractRightChild

For the remainder of this section, we prove that Algorithm 2 (ExtractRightChild) is equivalent to and faster than Algorithm 3 (ExtractRaw). For the following lemmas, let K be a simplicial complex, let $f_0 \colon K_0 \to \mathbb{R}$ be an injective function, and let (H, T, C, m) be the output of ExtractRightChild(K, f_0).

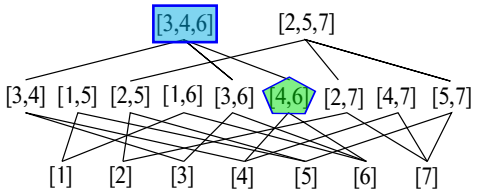
First, we show that (H, T, C, m) is a partition of K.

Lemma 5 (Partition) *The sets H, T, and C partition K.*

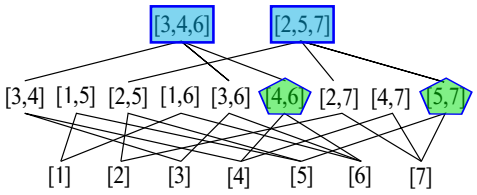
Proof. By Line 3 and Line 4 of Algorithm 2, ExtractRightChild iterates over all $\sigma^d \in K$ with $d > 0$ once. Each σ is either assigned or unassigned. If σ is unassigned, there are two options; σ may be a left-right parent, or it may not be. If σ is a left-right parent, Line 8 ensures that σ is put into H. Otherwise, Line 12 ensures that σ is put into C. If σ is assigned, then σ was assigned to T in Line 8. Thus, every $\sigma^d \in K$ with $d > 0$ must be assigned to exactly one of H, T, or C. Then, every σ^0 is again either assigned or unassigned. If assigned, $\sigma^0 \in T$. If unassigned, σ^0 is added to C in Line 14. Thus, every $\sigma \in K$ is assigned one of H, T, or C, making H, T, and C partition K. □

We will show that (H, T, C, m) satisfies (i), (ii), and (iii) of Theorem 3. Later in this section, we show that any GVF with these properties is unique.

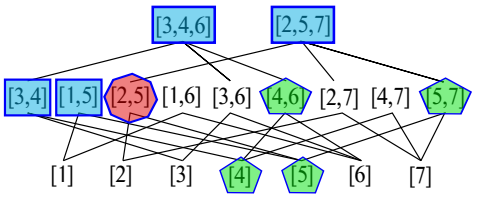
First, we show (iii) and one direction of (ii).



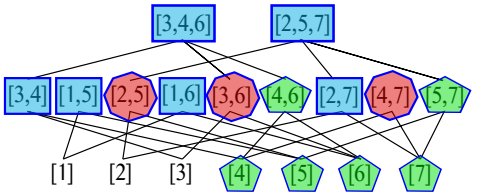
(a) The simplex $[3, 4, 6]$ is a left-right parent because $\ell \circ \rho([3, 4, 6]) = \ell(4, 6) = [3, 4, 6]$. The algorithm adds $[3, 4, 6]$ to H and $[4, 6]$ to T.



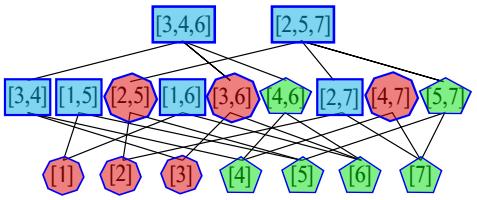
(b) The simplex $[2, 5, 7]$ is also a left-right parent. The algorithm adds $[2, 5, 7]$ to H and $[5, 7]$ to T.



(c) Both $[3, 4]$ and $[1, 5]$ are left-right parents. The simplex $[2, 5]$ is not a left-right parent because $\ell \circ \rho([2, 5]) = [1, 5] \neq [2, 5]$. The algorithm adds $[2, 5]$ to C.



(d) The loop in Line 3 is complete.



(e) The algorithm adds unassigned vertices to C.

Figure 2: Here we see a visualization of Algorithm 2, ExtractRightChild on the complex shown in Figure 1. Algorithm 2 partitions the nodes of the Hasse diagram into three sets, H, T and C. Elements of H are represented by blue rectangles, elements of T by green pentagons and elements of C by red hexagons.

Lemma 6 (Child Heads are Parents) *Let $\sigma \in H$. Then, σ is a left-right parent and $m(\rho(\sigma)) = \sigma$.*

Proof. Recall that H is the second output of ExtractRightChild(K,f_0), given in Algorithm 2. As Line 8 is the only step in which simplices are added to H and is within an if statement that checks if σ is a left-right parent, σ must be a left-right parent. Also within the if statement, Line 9 adds $(\rho(\sigma), \sigma)$ to m, which means that $m(\rho(\sigma)) = \sigma$. □

Now we show the reverse direction of (ii).

Lemma 7 (Child Parents are Heads) *Let $\sigma \in K$. If σ is a left-right parent, then $\sigma \in H$.*

Proof. Recall that in order for σ to be a left-right parent, we must have $\ell(\rho(\sigma)) = \sigma$. Now, we consider two cases. For the first case, suppose $\sigma \in C$. Then σ is added to C in Line 12 of Algorithm 2 when $\rho(\sigma)$ must already be assigned to $h <_{lex} \sigma$. So, $\ell(\rho(\sigma)) = h \neq \sigma$ and σ is not a left-right parent.

For the second case, suppose $\sigma = [v_0, v_1, \ldots, v_d] \in T$. Then σ is added to T in Line 8 of Algorithm 2 where $\sigma = \rho(h)$ for some $h = [v_{-1}, v_0, \ldots, v_d] \in H$ with $f_0(v_{-1}) < f_0(v_0)$. Notice that $\xi = [v_{-1}, v_1, v_2, \ldots, v_d]$ is a face of h and $\xi <_{lex} \sigma$. Then, $\ell(\rho(\sigma)) = \ell([v_1, \ldots, v_d]) \leq_{lex} [v_{-1}, v_1, v_2, \ldots, v_d] <_{lex} [v_0, v_1, \ldots, v_d] = \sigma$ and σ is not a left-right parent.

Thus, if σ is a left-right parent, then $\sigma \in H$.

□

To see (H, T, C, m) satisfies (i) we have the following lemma:

Lemma 8 (Consistency) *The tuple (H, T, C, m) is a gradient vector field consistent with f_0.*

Proof. Let $\varepsilon > 0$ and $d = \dim(K)$. Let $(H, T, C, m) =$ ExtractRightChild(K,f_0). We define

$$\delta := \min\{\varepsilon, \min_{v,w \in K_0} |f(v) - f(w)|\}.$$

We define $f\colon K \to \mathbb{R}$ recursively as follows: for all vertices $v \in K_0$, define $f(v) := f_0(v)$. Now, assume that f is defined on the i-simplices, for some $i \geq 0$. For each $\sigma \in K_{i+1}$, we initially assign $f(\sigma) = \max_{\tau \prec \sigma} f(\tau)$, then we update:

$$f(\sigma) = f(\sigma) + \begin{cases} -2^{1-2i}\delta & \text{if } \sigma \text{ is a left-right parent;} \\ 2^{1-2i}\delta & \text{otherwise,} \end{cases} \tag{1}$$

where j is the index of σ in the lexicographic ordering of all simplices. We make one final update:

$$f(\sigma) = f(\sigma) + \begin{cases} 2^{-2i}\delta & \text{if } \sigma \text{ is a left-right child;} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

We need to show that (H, T, C, m) and f satisfy the three properties in Definition 2.

First, we show Part (a) of Definition 2 holds for f as defined above (that (H, T, C, m) is the GVF corresponding to f). Let (H', T', C', m') be the GVF corresponding to f. Since H, T, C partitions K by Lemma 5, it suffices to show that m is a bijection and $m = m'$. The only time that simplices are added to H or T happens directly alongside when pairs are added to m in lines 8 and 9, forcing that m must be a match.

Let $(\tau, \sigma) \in m$. Let $i = \dim(\sigma)$. By Lemma 7, σ is a left-right parent and $\tau = \rho(\sigma)$, which means that (τ, σ) is a left-right pair. We follow the computation of $f(\sigma)$. Since (τ, σ) is a left-right pair, τ is the rightmost face of σ, which means $f(\sigma)$ is initialized to $f(\tau)$. Since σ is a left-right parent, $f(\sigma)$ is updated by (1) to $f(\sigma) = f(\sigma) - 2^{1-2i}\delta$. Since σ is not a left-right child, nothing changes in (2). Thus, $f(\sigma) < f(\tau)$. Next, let $\tau' \prec \sigma$ such that $\tau' \neq \tau$ and $\dim(\tau') = i - 1$. We follow the computation of $f(\tau')$. Since τ is the only face of σ that is a left-right child, for any other $\tau' \prec \sigma$, (2), adds zero to the definition of $f(\tau')$. Recalling that (2) adds $2^{-2(i-1)}\delta$ to the definition of $f(\tau)$, we find that $f(\tau) \geq f(\tau') + 2^{-2(i-1)}\delta$, and

$$f(\sigma) = f(\tau) - 2^{1-2i}\delta \geq f(\tau') + 2^{-2(i-1)}\delta - 2^{1-2i}\delta \geq f(\tau').$$

Because σ may be any arbitrary left-right parent, we can guarantee that the above inequality is valid for any $(\sigma, \tau) \in m$ when related to any other faces of σ. Thus, f is discrete Morse, since it is impossible for f to violate the inequality given in Definition 1.

Since $f(\tau) > f(\sigma)$ and f is a discrete Morse function, we obtain $(\tau, \sigma) \in m'$. Each of these statements are biconditional, so we have shown that $m = m'$.

Part (b) of Definition 2 ($f|_{K_0} = f_0$) holds trivially.

Finally, we show Part (c) of Definition 2 holds (that $|f(\sigma) - \max_{v \in \sigma} f_0(v)| \leq \varepsilon$). By construction,

$$|f(\sigma - \max_{v \in \sigma} f_0(v)| \leq \left(\sum_{i=1}^{d} 2^{-i}\right)\delta = (1 - 2^{-d})\delta < \varepsilon.$$

□

Properties (i), (ii), and (iii) are quite restrictive. In fact, they uniquely determine a GVF, as we now show.

Theorem 9 (Unique GVF) *Let K be a simplicial complex and let $f_0\colon K_0 \to \mathbb{R}$ be an injective function. There is exactly one gradient vector field, (H, T, C, m), with the following two properties:*

(i) (H, T, C, m) is consistent with f_0.

(ii) For all $\sigma \in K$, $\sigma \in H$ if and only if σ is a left-right parent.

(iii) For all $\sigma \in H$, $m(\rho(\sigma)) = \sigma$.

Proof. Let K and f_0 be as defined in the theorem statement. Let $\tilde{f}\colon K \to \mathbb{R}$ be defined for each simplex $\sigma \in K$ by $\tilde{f}(\sigma) := \max_{v\in\sigma} f_0(v)$. Let (H, T, C, m) and (H', T', C', m') be two GVFs that satisfy (i), (ii), and (iii).

Let $\sigma \in H$. By the forward direction of (ii), we know that σ is a left-right parent. By the backward direction of (ii), we know that $\sigma \in H'$. Thus, we have shown that $H \subseteq H'$. Repeating this argument by swapping the roles of H and H' gives us $H' \subseteq H$.

Since $\sigma \in H = H'$ and because (iii) holds, we have shown that σ is paired with $\rho(\sigma)$ in both matchings, and specifically $m(\rho(\sigma)) = \sigma = m'(\rho(\sigma))$. Since m and m' are bijections by (i), we also know that:

$$T = \{\tau \in K \mid \exists \sigma \in H \text{ s.t. } m(\rho(\sigma)) = \sigma\} = T'.$$

Thus, $T = T'$ and $m = m'$.

Finally, we conclude:

$$C = K \setminus (T \cup H) = K \setminus (T' \cup H') = C',$$

which means that (H, T, C, m) and (H', T', C', m') are the same GVF. Thus, we conclude that the gradient vector field satisfying (i), (ii), and (iii) is unique. □

Since ExtractRightChild and ExtractRaw both satisfy the hypothesis of Theorem 9, the outputs of the algorithms must be the same.

Theorem 10 (Algorithm Equivalence) *Let K be a simplicial complex and let $f_0\colon K_0 \to \mathbb{R}$ be an injective function. Then* ExtractRaw*(K, f_0) and* ExtractRightChild*(K, f_0) yield identical outputs.*

Proof. By Theorem 3 and Lemma 12, the output of ExtractRaw satisfies the properties in Theorem 9. By Lemma 8, Lemma 6, and Lemma 7, the output of ExtractRightChild satisfies the properties of Theorem 9. Then, by Theorem 9, ExtractRaw and ExtractRightChild are equivalent.

□

When we consider the runtime and space usage of ExtractRightChild, we find the following:

Theorem 11 (New Runtime) *Given a simplicial complex K (represented as a Hasse diagram), and an injective function $f_0 : K_0 \to \mathbb{R}$,* ExtractRightChild *computes a GVF consistent with f_0 in $O(dn)$ time and uses $O(n)$ space.*

Proof. First, line Line 1 decorates the Hasse diagram. By Lemma 4, the decoration takes $O(dn)$ time and $O(n)$ space. Lines 3-13, process each node of the decorated Hasse diagram. Each iteration of the loop is $O(1)$ in time and space because all required data was computed while decorating. As there are $n - n_0$ nodes to process, Lines 3-13 takes $O(n)$ time and uses $O(1)$ space. Finally, we iterate over the zero-simplices in $O(n_0)$ time.

The bottleneck of space and time usage of the algorithm is decorating the Hasse diagram, therefore, the algorithm takes $O(dn)$ time and $O(n)$ space. □

5 Discussion

In this paper, we identified properties of the Extract and ExtractRaw algorithms [13]. We used these properties to simplify ExtractRaw to the equivalent algorithm ExtractRightChild. Our simplification improves the runtime from $\Omega(n^2 \log n)$ to $O(dn)$.

There are several possible extensions of this work. The problem of finding tight bounds on the runtime of Extract is interesting and open. We plan to implement our approach on high dimensional data sets, and to further improve to the runtime. We intend to explore a cancellation algorithm that performs the same task as ExtractCancel, eliminating critical pairs with small persistence. Our conjectured cancellation algorithm iterates over critical simplices and applies ExtractRightChild.

Constructing Morse functions that do not require preassigned function values on the vertices is a related area of active research. The problem of finding a Morse function with a minimum number of critical simplices is NP-hard [12]. In [3], Bauer and Rathod show that for a simplicial complex of dimension $d \geq 3$ with n simplices, it is NP-hard to approximate a Morse matching with a minimum number of critical simplices within a factor of $O(n^{1-\epsilon})$, for any $\epsilon > 0$. The question is open for 2-dimensional simplicial complexes.

Acknowledgements This material is based upon work supported by the National Science Foundation under the following grants: CCF 1618605 & DMS 1854336 (BTF) and DBI 1661530 (DLM). Additionally, BH thanks the Montana State Undergraduate Scholars Program. All authors thank Nick Scoville for introducing us to KKM [13] and for his thoughtful discussions.

References

[1] U. Bauer. *Persistence in Discrete Morse Theory.* PhD thesis, Niedersächsische Staats-und Universitätsbibliothek Göttingen, 2011.

[2] U. Bauer, C. Lange, and M. Wardetzky. Optimal topological simplification of discrete functions on surfaces. *Discrete and Computational Geometry*, 47(2):347–377, 2012.

[3] U. Bauer and A. Rathod. Hardness of approximation for Morse matching. arXiv:1801.08380, 2018.

[4] L. Čomić and L. De Floriani. Dimension-independent simplification and refinement of Morse complexes. *Graphical Models*, 73(5):261–285, 2011.

[5] T. Dey, J. Wang, and Y. Wang. Graph reconstruction by discrete Morse theory. In *34th Symposium on Computational Geometry (SoCG)*, pages 31:1–31–13, 2018.

[6] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[7] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete and Computational Geometry*, 30(1):87–107, 2003.

[8] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28:511–533, 2002.

[9] R. Forman. Discrete Morse theory for cell complexes. *Advances in Mathematics*, 134:90–145, 1998.

[10] R. Forman. A user's guide to discrete Morse theory. *Séminaire Lotharingien de Combinatoire*, 42:Art. B48c, 35pp, 2002.

[11] P. Hersh. On optimizing discrete Morse functions. *Advances in Applied Math*, 35:294–322, 2005.

[12] M. Joswig and M. Pfetsch. Computing optimal Morse matchings. *SIAM Journal on Discrete Mathematics (SIDMA)*, 20(1):11–25, 2006.

[13] H. King, K. Knudson, and N. Mramor. Generating discrete Morse functions from point data. *Experimental Mathematics*, 14:435–444, 2005. MR2193806.

[14] K. Knudson. *Morse Theory: Smooth and Discrete*. World Scientific Publishing Company, 2015.

[15] T. Lewiner, H. Lopes, and G. Tavares. Toward optimality in discrete Morse theory. *Experimental Mathematics*, 12:271–285, 2003.

[16] J. Milnor. *Morse Theory*. Princeton University Press, Princeton, New Jersey, 1963.

[17] K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Discrete and Computational Geometry*, (50):330, 2013.

[18] R. Raz. On the complexity of matrix product. *SIAM Journal on Computing*, 32:1356–1369, 2003.

[19] N. Scoville. *Discrete Morse Theory*. American Mathematical Society, Providence, Rhode Island, 2019.

A Additional Details for Extract

To put our result in context, we now provide a glimpse into the inner workings of Extract, and reveal the underlying properties of ExtractRaw which give it an identical output to ExtractRightChild. We also provide a formal runtime analysis of ExtractRaw to verify that ExtractRightChild provides an improved time complexity.

A.1 Subroutines for Extract

In this section, we recall the algorithms proposed by KKM [13]. Note that we made some slight modifications to the presentation of KKM's initial description to improve readability. The modifications do not affect the asymptotic time or space used by the algorithm, although it does remove some redundant computation.

In particular, we modified the inputs to explicitly pass around a GVF so that the inputs of each algorithm are clear. We simplified notation and inlined the subroutine Cancel. From the previous modifications, we observed that the algorithm recomputes a gradient path that is currently in scope and so we simply unpack the path on Line 11 of Algorithm 4.

ExtractRaw computes the lower link of each vertex v in a simplicial complex, and assigns $v \in C$ if $\text{lowerlink}_K(v) = \emptyset$. If $\text{lowerlink}_K(v) \neq \emptyset$, its lower link is recursively inputted into ExtractRaw and this recursion continues until an empty lower link is reached. When the lower link is not empty, ExtractRaw assigns $v \in T$ and the smallest function valued vertex ω_0 in $\text{lowerlink}_K(v)$ is combined with v and added to H, carrying with this assignment a mapping m from $\omega_0 * v \in H$ to $v \in T$. As the recursion continues, higher dimensional simplices in the lower start of v are able to be assigned to both H and T based on combinations consistent with the assignments of the vertices and the original mappings of m. Higher dimensional critical cells are assigned similarly by combining the current vertex and each previously computed $\sigma \in C$ from the last recursion, until all simplices have been assigned.

Then, because ExtractRaw may have extraneous critical cells, Cancel works to reduce the number of critical cells by locating "redundant" gradient paths to a critical simplex and reversing them after the first pass by ExtractRaw, refining the output of Extract .

Algorithm 3 [13] ExtractRaw

Input: simp. complx. K, injective fcn. $f_0 : K_0 \to \mathbb{R}$
Output: a GVF consistent with f_0

1: $T \leftarrow \emptyset$, $H \leftarrow \emptyset$, $C \leftarrow \emptyset$, $m \leftarrow \emptyset$
2: **for all** $v \in K$ **do**
3: Let K' := the lower link of v.
4: **if** $K' = \emptyset$ **then**
5: Add v to C
6: **else**
7: Add v to T
8: $(T', H', C', m') \leftarrow$ Extract(K', f_0, ∞)
9: $w_0 \leftarrow arg\ \min_{w \in C'_0}\{f_0(w)\}$
10: Add $w_0 v$ to H
11: Define $m(w_0 * v) := v$
12: For each $\sigma \in C' \setminus \{w_0\}$, add $v * \sigma$ to C
13: **for all** $\sigma \in T'$ **do**
14: Add $v * \sigma$ to T
15: Add $v * m'(\sigma)$ to H
16: Define $m(v * \sigma) = v * m'(\sigma)$
17: **return** (H, T, C, m)

Let $p \in \mathbb{N}$, $\sigma \in K_p$ be a critical simplex. Let $\mathcal{G}^j_\sigma$

Algorithm 4 [13] ExtractCancel

Input: simplicial complex K, injective function $f_0 \colon K_0 \to \mathbb{R}$, $p \geq 0$, $j \in \mathbb{N}$, and GVF γ
Output: Gradient vector field on K

```
1: Let (H, T, C, m) be the four components of γ
2: for all σ ∈ C_j do
3:     s ← max_{v∈σ} f_0(v)
4:     S ← {Γ | Γ ∈ G^j_σ, s − max_{w∈Γ_L} f_0(w) < p}
5:     for all Γ ∈ S do
6:         m_Γ ← ∞
7:         if Γ_L ≠ Γ'_L for any other Γ' ∈ S then
8:             m_Γ ← max_{w∈Γ_L} f_0(w)
9:     Γ* ← arg min_{Γ∈S}{m_Γ}
10:    if m_{Γ*} ≠ ∞ then
11:        {σ_1, τ_1, ··· , σ_k, τ_k} ← Γ*
12:        Remove τ_k, σ_1 from C
13:        Add τ_k to T; Add σ_1 to H
14:        Add (τ_k, σ_k) to m
15:        for i = 1, ..., k − 1 do
16:            Remove (τ_i, σ_{i+1}) from m
17:            Add (τ_i, σ_i) to m
18: return (H, T, C, m)
```

denote the set of all nontrival gradient path starting at $\sigma \in C_j$ and ending in C_{j-1}.

A.2 Analysis of ExtractRaw

In this appendix, we provide the analysis Algorithm 1 from Section 3. In what follows, let K be a simplicial complex and let $f_0 \colon K_0 \to \mathbb{R}$ be an injective function.

Lemma 12 (Raw Heads are Parents) *Let (H, T, C, m) be the output of* ExtractRaw(K, f_0). *Every simplex in H is a left-right parent. Furthermore, for all $\sigma \in H$, $m(\rho(\sigma)) = \sigma$.*

Proof. Let $\sigma \in H$. We show that σ is a left-right parent by induction on the dimension of K. When $\dim(K) = 1$, σ is an edge, and $\sigma = m(\tau)$ for some vertex $\tau \in T$. In Line 10 of Algorithm 3 σ is defined as $m(\tau) = [w_0, \tau]$ where $w_0 \in C'_0$ so that $f_0(w_0)$ is smallest. So, σ is a left-right parent. Furthermore, $m(\rho(\sigma)) = m(\rho([w_0, \tau])) = m(\tau) = \sigma$.

Suppose every $\sigma \in H$ is a left-right parent when $\dim(K) \leq d$ and consider $\dim(K) = d + 1$. If σ is a $(d+1)-$simplex, $\sigma = m(\tau)$ is defined in Line 15 of Algorithm 3, when a vertex v is selected in Line 2 of Algorithm 3. We extend the GVF on the $\text{lowerlink}_K(v)$ to include the lower star of v. We have $m(\tau) = v * m'(\alpha)$ where $\alpha = [v_1, \ldots, v_d] \in T'$, $m'(\alpha) = [v_0, v_1 \ldots v_d] \in H'$. Since α and $m'(\alpha)$ are in the $\text{lowerlink}_K(v)$ we have $f(v_i) < f(v)$ for $0 \leq i \leq d$. Then $\tau = v * \alpha = [v_1, \ldots, v_d, v]$ and $\sigma = m(\tau) = [v_0, v_1, \ldots, v_d, v]$.

By the induction hypothesis $m'(\alpha) \in H'$ is a left-right parent. If σ is not a left-right parent, we can remove v from σ and τ and contradict that $m'(\alpha) \in H'$.

Furthermore, $m(\rho(\sigma)) = m(\rho(m(\tau))) = m(\rho([v_0, v_1, \ldots, v_d, v])) = m([v_1, \ldots, v_d, v]) = m(\tau) = \sigma$. This proves the claim. □

Lemma 13 (Raw Parents are Heads) *Let (H, T, C, m) be the output of* ExtractRaw(K, f_0). *Let $\sigma \in K$. If σ is a left-right parent, then $\sigma \in H$.*

Proof. We show if $\sigma \in T \cup C$ then σ is not a left-right parent. First, suppose $\sigma \in T$. We use induction on $\dim(K)$ to show σ is not a left-right parent. For the base case, $\dim(K) = 1, \sigma$ is a vertex and can not be a left-right parent.

Suppose $\sigma \in T$ is not a left-right parent when $\dim(K) \leq d$ and consider $\dim(K) = d + 1$. Then σ is added to T in Line 14 of Algorithm 3 when a vertex v is selected in Line 2. As in Lemma 12, write $\sigma = v * \alpha = [v_1, \ldots, v_d, v]$ for some $\alpha \in T'$.

By the induction hypothesis α is not a left-right parent, thus $\ell \circ \rho(\alpha) \neq \alpha$, and there exists a vertex v_{-1} such that $\ell(\rho(\alpha)) = [v_{-1}, v_2, v_3 \ldots, v_d]$ where $f(v_{-1}) < f(v_1)$. We have $\ell \circ \rho(\sigma) = \ell \circ \rho([v_1, v_2, \ldots, v_d, v]) = \ell([v_2, v_3 \ldots, v_d, v]) = [v_{-1}, v_2, \ldots, v_d, v] \neq \sigma$.

Now, suppose $\sigma \in C$. There are two places where elements are added to C, Line 5 of Algorithm 3 and Line 12. In Line 5 c is a vertex and can not be a left-right parent.

In Line 12 c is defined as $c = v * \alpha$ for some $\alpha = [v_0, v_1, \ldots, v_d] \in C' \backslash w_0$ where $w_0 \in C'_0$ so that $f_0(w_0)$ is smallest. Now $\ell \circ g(c) = \ell \circ g([v_0, v_1, \ldots, v_d, v]) = \ell([v_1, \ldots, v_d, v]) = [w_0, v_1, \ldots, v_d, v] \neq c$. We have shown that if σ is a left-right parent, then $\sigma \in H$.

□

We summarize the properties of ExtractRaw in the following theorem.

Theorem 3 (Properies of ExtractRaw) *Let K be a simplicial complex, let $f_0 \colon K_0 \to \mathbb{R}$ be an injective function, and suppose (H, T, C, m) is the output of* ExtractRaw(K, f_0). *Let $\varepsilon > 0$. Then, there exists a discrete Morse function $f \colon K \to \mathbb{R}$ such that the following hold:*

(i) (H, T, C, m) is a GVF consistent with f_0.

(ii) Let $\sigma \in K$. Then, $\sigma \in H$ if and only if σ is a left-right parent.

(iii) For all $\sigma \in H$, $m(\rho(\sigma)) = \sigma$.

(iv) The runtime of ExtractRaw *is $\Omega(n^2 \log n)$.*

Proof. (i) is proven in Theorem 3.1 of [13]. By Lemma 12 and Lemma 13, we conclude (ii). Also by Lemma 12 we can guarantee (iii).

To show (iv), we observe that the worst-case runtime for a single execution of Line 8 of Algorithm 3 happens when the lower link of v is of size $\Theta(n/2)$. Computing the optimal pairings that ExtractRaw returns is at least as hard as computing the homology of K', which is of the time complexity of matrix multiplication. By [18], we know that the runtime of ExtractRaw is lower-bounded by $\Omega(n^2 \log n)$. □

A Simple Algorithm for kNN Sampling in General Metrics*

Kirk P. Gardner† and Donald R. Sheehy‡

Abstract

Finding the kth nearest neighbor to a query point is a ubiquitous operation in many types of metric computations, especially those in unsupervised machine learning. In many such cases, the distance to k sample points is used as an estimate of the local density of the sample. In this paper, we give an algorithm that takes a finite metric $(P, \mathbf{d})$ and an integer k and produces a subset $M \subseteq P$ with the property that for any $q \in P$, the distance to the second nearest point of M to q is a constant factor approximation to the distance to the kth nearest point of P to q. Thus, the sample M may be used in lieu of P. In addition to being much smaller than P, the distance queries on M only require finding the second nearest neighbor instead of the kth nearest neighbor. This is a significant improvement, especially because theoretical guarantees on kth nearest neighbor methods often require k to grow as a function of the input size n.

1 Introduction

Subsampling is a fundamental step in many large scale data analysis problems. The goal is that the distribution of the subsample $M \subset P$ resembles the distribution on the whole data set, P. On the one hand, preserving purely statistical properties is often achieved by random sampling. This paper, on the other hand, considers preserving metric properties of a subsample. Specifically, we want a sample where the distance to the second nearest neighbor in M is approximately the distance to the kth nearest neighbor in P. We call this a *kth nearest neighbor sample* and it balances between competing demands of representing the underlying distribution and the underlying metric.

Figure 3 shows a kth nearest neighbor sample of a collection of points in the plane. The points are 5 times denser on the right half and there are correspondingly more points in the sample on that side. Figure 1 shows a random sample of the same point set with the same number of points sampled. The random sample also has more points on the denser half, but it has more variability in the distance between samples; some are virtually on top of each other. The greedy sample (Figure 2) is a standard way to produce a uniform sample at a particular scale, but that scale does not vary with the density. In this sense, the kNN sample achieves a balance between the random sample and the uniform (i.e. greedy) sample.

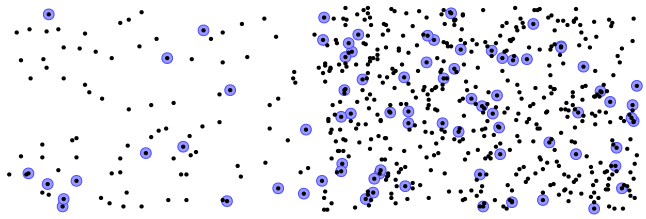


Figure 1: A random sample.

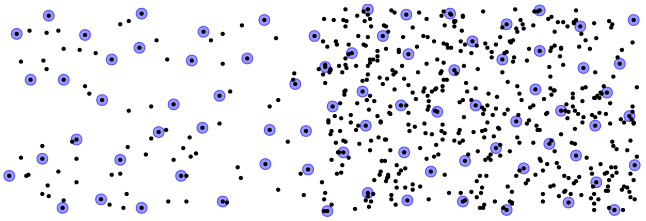


Figure 2: A greedy sample.

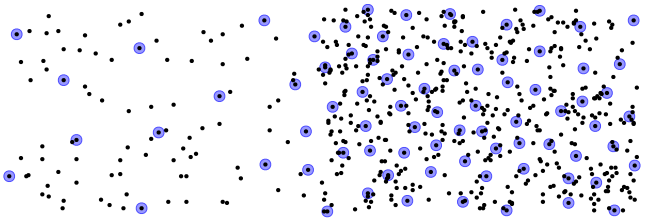


Figure 3: A kth nearest neighbor sample.

In prior work [7], we showed how a variation of Delaunay refinement can be used in the plane to compute such a sample. In this paper, we prove that a simple algorithm can compute a kNN sample in any metric space. Then, we show how data structures and ideas from nearest neighbor search can be adapted to speed up the computation.

The algorithm is presented in its simplest form in Section 3. There, we prove upper and lower bounds on the kth nearest neighbor distance in terms of the

*This work was partially supported by the NSF under grants CCF-1464379, CCF-1525978, and CCF-1652218.

†Department of Computer Science, North Carolina State University, kpgardn2@ncsu.edu

‡Department of Computer Science, North Carolina State University, don.r.sheehy@gmail.com

2nd nearest neighbor distance in the sample. Then, in Section 4, we describe a neighborhood graph structure adapted from Clarkson [3] that we use to speed up the local search step of the algorithm. In Section 5, we put all these pieces together to bound the overall running time. We report on our open source implementation and give some demonstrations of the code in action in Section 7. Finally, in Section 8, we propose some open problems that remain.

1.1 Related Work

The distance to the kth nearest neighbor has been used for a long time as a density estimator (see Biau and Devroye [1]). It has also been used in pointwise estimates of the local density in metric measure spaces (see Cutler and Dawson [5] and the survey by Clarkson [4]). There are many data structures that compute kth nearest neighbors efficiently, perhaps the fastest in practice are the Faster Cover Trees of Izbicki and Shelton [11]. There has also been theoretical work by Har-Peled and Raichel on a general framework for computing aggregate statistics like the kth nearest neighbor distance with the so-called Net and Prune paradigm [10]. The first algorithms for computing kNN samples was the work of Gardner and Sheehy using Delaunay refinement, but was limited to Euclidean space [7]. The main algorithmic paradigm we use here is based on greedy orderings, which have been used since the 1980's for approximate k-center clustering [6, 8]. The underlying data structure is a variant of the *sb* structure of Clarkson [3, 2], which was first analyzed for greedy orderings by Har-Peled and Mendel [9].

2 Background

We will deal with finite subsets of metric spaces $(X, \mathbf{d})$, where the X is the set of points and $\mathbf{d}$ is the metric. For $x \in X$ and subsets S of X, we define

$$\mathbf{d}(x, S) := \min_{s \in S} \mathbf{d}(x, s).$$

The minimum distance to k points in S is denoted

$$\mathbf{d}_{S,k}(x) := \min_{U \in \binom{S}{k}} \max_{y \in U} \mathbf{d}(x, y).$$

In particular, $\mathbf{d}_{S,1}(x) = \mathbf{d}(x, S)$. The *Hausdorff distance*, $\mathbf{d}_H$, is defined on subsets of X as

$$\mathbf{d}_H(A, B) := \max\{\max_{a \in A} \mathbf{d}(a, B), \max_{b \in B} \mathbf{d}(b, A)\}.$$

A *metric ball in* S is the set of points of S within a fixed radius of a point in S. A *minimum r-cover* of a set S is the smallest set of centers of metric balls of radius r whose union contains S. Equivalently, it is the smallest subset $C \subset S$ such that $\mathbf{d}_H(S, C) \leq r$.

The *doubling constant* of a metric is the size of the largest minimum r-cover of any ball of radius $2r$. The *doubling dimension* is the base-two logarithm of the doubling constant. A bound on the doubling dimension allows one to apply the kind of packing arguments as are often used in Euclidean space.

The *spread* $\Delta(P)$ of a finite metric P is the ratio of the largest to smallest pairwise distances. The *k-spread* is the ratio

$$\Delta_k(P) := \frac{\max_{p,q \in P} \mathbf{d}(p, q)}{\min_{p \in P} \mathbf{d}_{P,k}(p)}.$$

A set with spread Δ in a metric with doubling dimension d has at most $O(\Delta^d)$ points [9].

An *r-packing* is a set of points for which the minimum pairwise distance is at least r. An r-packing that is also an r-cover is called an *r-net*.

For a metric space X and a subset P, an *(α, β)-kNN sample* of P is a subset $M \subseteq P$ with the property that for all $x \in X$,

$$\alpha \mathbf{d}_{P,k}(x) \leq \mathbf{d}_{M,2}(x) \leq \beta \mathbf{d}_{P,k}(x).$$

We will refer to it simply as a kNN sample when the values of α and β are not important. The algorithm in this paper produces a $(1/5, 2)$-kNN sample.

Let $P = (p_1, \ldots, p_n)$ be an ordered subset of X. Let $P_i = \{p_1, \ldots, p_i\}$ be the ith *prefix*. The ordering is *greedy* if for all $i \in 2, \ldots, n$, we have

$$\mathbf{d}(p_{i+1}, P_i) := \mathbf{d}_H(P, P_i).$$

In other words, every point p_{i+1} is the farthest point from P_i. We will compute kNN-samples in a greedy order. Both the data structure in Section 4 and the algorithm in Section 5 will depend on this ordering for their correctness and efficient running time.

3 A Simple Algorithm

We present a simple algorithm that generates an (α, β)-kNN-sample as a subset of the input. It iteratively builds M from P by adding a point p of P to M as long as $\mathbf{d}_{P,k}(p) \leq 2\mathbf{d}_{M,1}(p)$. In this section, we prove that any such algorithm produces a $(1/5, 2)$-kNN-sample.

Algorithm 1 kNNSample(P, k)

```
M ← ∅
while ∃p ∈ P such that d_{P,k}(p) ≤ 2d_{M,1}(p) do
    Add p to M
return M
```

Theorem 1 *Let P be a subset of a metric space X. Let $M =$ kNNSample(P, k) for some $k \geq 2$. Then,*

$$\frac{1}{5}\mathbf{d}_{P,k} \leq \mathbf{d}_{M,2} \leq 2\mathbf{d}_{P,k}.$$

Proof. We first prove the lower bound on $\mathbf{d}_{M,2}$. Let $x \in P$ be any point. Let v_1 and v_2 be the two nearest points to x in M, so,

$$\mathbf{d}_{M,1}(x) = \mathbf{d}(x, v_1) \leq \mathbf{d}(x, v_2) = \mathbf{d}_{M,2}(x). \tag{1}$$

Let $v \in \{v_1, v_2\}$ be whichever point was added later by the algorithm, which guarantees that

$$\mathbf{d}_{P,k}(v) \leq 2\mathbf{d}(v_1, v_2). \tag{2}$$

We can now bound $\mathbf{d}_{P,k}(x)$ as follows.

$$\begin{aligned}
\mathbf{d}_{P,k}(x) &\leq \mathbf{d}_{P,k}(v) + \mathbf{d}(v, x) && [\mathbf{d}_{P,k} \text{ is 1-Lipschitz}] \\
&\leq 2\mathbf{d}(v_1, v_2) + \mathbf{d}(v, x) && [\text{by (2)}] \\
&\leq 2(\mathbf{d}(v_1, x) + \mathbf{d}(x, v_2)) + \mathbf{d}(v, x) && [\text{triangle inequality}] \\
&\leq 5\mathbf{d}_{M,2}(x) && [\text{by (1)}].
\end{aligned}$$

Now, we prove the upper bound on $\mathbf{d}_{M,2}$. Suppose for contradiction that there exists a point $x \in X$ such that $\mathbf{d}_{M,2}(x) > 2\mathbf{d}_{P,k}(x)$. Let S be the k closest points in P to x. Let r be the radius of the minimum enclosing ball of S and note that $r \leq \mathbf{d}_{P,k}(x)$. If $\mathbf{d}_{M,1}(s) < r$ for some $s \in S$, then let $m \in M$ be the nearest neighbor of s in M. It follows from the triangle inequality that $\mathbf{d}(x, m) \leq \mathbf{d}(x, s) + \mathbf{d}(s, m) \leq 2\mathbf{d}_{P,k}(x) < \mathbf{d}_{M,2}(x)$. There can be only one point $m \in M$ whose distance to x is less than $\mathbf{d}_{M,2}(x)$. However, if $\mathbf{d}_{M,1}(s) < r$ for all $s \in S$, then there would be a smaller minimum enclosing ball for S centered at m, contradicting our choice of r. So, for at least one point $s_* \in S$, we have $\mathbf{d}_{M,1}(s_*) \geq r$. Therefore, by the triangle inequality, $\mathbf{d}_{P,k}(s_*) \leq 2r \leq 2\mathbf{d}_{M,1}(s_*)$. The existence of such a point would cause the algorithm to add s_* to M, contradicting the assumption that $M = \text{kNNSample}(P, k)$. Thus, we conclude that no such x exists and indeed $\mathbf{d}_{M,2} \leq 2\mathbf{d}_{P,k}$. □

In Sections 5 and 6, we explain how to efficiently test this condition and bound the running time. Efficiency is achieved by constructing the sample in a greedy order. Note that the simplified algorithm does not rely on a particular ordering.

4 A Cluster Graph

In this section we will define a graph on the a subset of points that can be used to rapidly shrink the search space when computing $\mathbf{d}_{P,k}$ in the middle of our algorithm. It is a variation on a data structure used in the construction of the *sb* data structure of Clarkson [3]. Each vertex in the graph will be a point that we have already added to our sample. Moreover, each vertex will track the uninserted points that are closest to it. Every time a new point is considered for addition, we use the vertices adjacent to its nearest neighbor to search for nearby points. The formal definition is given below.

Let P be a finite metric space, and let $M \subset P$ be any subset. The *cluster* of $x \in M$ is the set of points C_x in P that are closer to x than to any other point in M. We break ties arbitrarily but consistently. The clusters are discrete Voronoi cells.

The *cluster graph* G_M on vertex set M has points a and b adjacent if there exist $a' \in C_a$ and $b' \in C_b$ such that

$$\mathbf{d}(a', b') \leq 2\max\{\mathbf{d}(a', a), \mathbf{d}(b', b)\}.$$

If a and b are adjacent, we denote this as $a \sim b$. The graph has self loops at every vertex. Because $a' \in C_a$ and $b' \in C_b$, the adjacency condition is equivalent to

$$\mathbf{d}(a', b') \leq \max\{2\mathbf{d}_{M,1}(a'), 2\mathbf{d}_{M,1}(b')\}.$$

So, if we wanted to try to add a' to M, we could find all the points within distance $2\mathbf{d}_{M,1}(a')$ among the the clusters of the vertices adjacent to a.

Moreover, we use the cluster graph to efficiently maintain itself under insertions. That is, we can use the graph to quickly find the edges incident to a newly inserted point.

Lemma 2 *Let $M \subset P$ and let $M' = M \cup \{a'\}$ for $a' \in C_a$ and $a \in M$. If $a' \sim c$ in $G_{M'}$, then there exists $b \in M$ such that $a \sim b \sim c$ in G_M. In other words, the neighbors of a' will be found among the neighbors of neighbors of a.*

Proof. By the definition of adjacency in $G_{M'}$ there exist points $b' \in C_{a'}$ and $c' \in C_c$ such that

$$\begin{aligned}
\mathbf{d}(b', c') &\leq 2\max\{\mathbf{d}(b', a'), \mathbf{d}(c', c)\} \\
&\leq 2\max\{\mathbf{d}(b', b), \mathbf{d}(c', c)\},
\end{aligned}$$

where b is the nearest point to b' in M as illustrated in Figure 4. From this, it follows that $b \sim c$ in G_M. Next, observe that

$$\mathbf{d}(a', b') \leq \mathbf{d}(b, b') \leq 2\max\{\mathbf{d}(b, b'), \mathbf{d}(a, a')\}.$$

So, it immediately follows that $a \sim b$ in G_M. □

Lastly, we would prefer to avoid the extensive checking required to see if two points are adjacent. At first sight, it seems to require searching for a pair that are particularly close. Instead, we follow the example of Clarkson [3] and prove a sufficient condition for bounding the neighbors just by using the radii of the clusters and the triangle inequality.

For a point $x \in M$, define its radius to be

$$\text{rad}(x) = \max_{y \in C_x} \mathbf{d}(y, x).$$

In other words, it is the distance from x to the farthest point in its cluster.

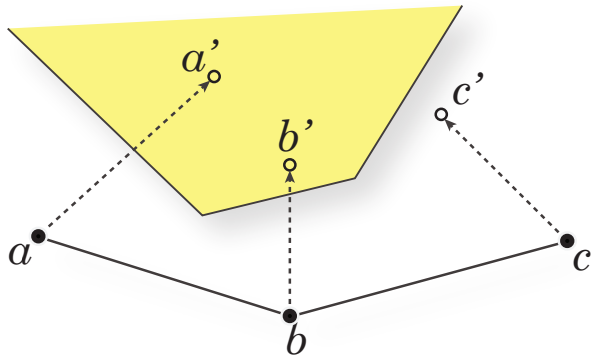


Figure 4: Here, $a' \in C_a$, $b' \in C_b$, and $c' \in C_c$. If adding a' would require creating an edge from a' to c, we will find c among the neighbors of neighbors of a. The cluster of a' is shown. The same pair (b', c') that witnesses a cluster graph edge from a' to c also guarantees the existence of the edge from b to c.

Lemma 3 *If $a \sim b$ in G_M, then*

$$\mathbf{d}(a,b) \leq \mathrm{rad}(a) + \mathrm{rad}(b) + 2\max\{\mathrm{rad}(a), \mathrm{rad}(b)\}.$$

Proof. If $a \sim b$, then there exist points $a' \in C_a$ and $b' \in C_b$ such that

$$\mathbf{d}(a',b') \leq 2\max\{\mathbf{d}(a,a'), \mathbf{d}(b,b')\}.$$

We simply observe that $\mathbf{d}(a,a') \leq \mathrm{rad}(a)$ and $\mathbf{d}(b,b') \leq \mathrm{rad}(b)$, so the result follows from the triangle inequality. □

Using the lemma above, we can quickly check if an edge ought to be removed from our cluster graph as new points are added and radii decrease. This distance condition is precisely what is needed to bound the space usage as long as the points are added in a greedy order.

Lemma 4 *If M is a prefix of a greedy ordering of $S \subseteq P$, then the cluster graph on M has maximum degree γ^3, where γ is the doubling constant of the metric.*

Proof. Let r be the maximum radius among all the clusters in the cluster graph. By Lemma 3, if $a \sim b$ in G_M, then $\mathbf{d}(a,b) \leq r_a + r_b + 2\max\{r_a, r_b\} \leq 4r$. So, the neighbors of any point $a \in M$ are contained in $\texttt{ball}(a, 4r)$. Because the points are added in a greedy order, no two points of M have distance less than r. By the definition of γ, the doubling constant, $\texttt{ball}(a, 4r)$ can be covered by γ^3 balls of radius $r/2$. Each such ball contains at most one point of M. Therefore, there are at most γ^3 neighbors. □

Corollary 5 *Updating the neighbors of the vertices in a cluster graph takes constant time per insertion in doubling metrics if one adds points in a greedy order.*

This last lemma and its corollary show the importance of using the greedy order. It makes the search for nearby neighbors efficient. The algorithm described in the following section will construct M in a greedy order. This greedy order is also important to the efficiency of the algorithm in other ways as we will see in the analysis.

5 Efficiently Computing the GreedykNN Algorithm

The biggest challenge in implementing the simple algorithm of Section 3 is that it requires computing or at least bounding $\mathbf{d}_{M,2}$ and $\mathbf{d}_{P,k}$ for every point.

As explained in Section 4, the main data structure is a kind of discrete Voronoi diagram. For each inserted point p, it has two parts: first, it stores the cluster C_p; second, it stores the neighbors of p, the other inserted points within some distance. This data structure will be used to guarantee that the points in the output are discovered in a greedy ordering.

The efficiency improvements in the algorithm over a linear search are achieved by only searching locally in the cluster graph. Thus, to prove the algorithm is correct, we need to show that it is sufficient to only search the neighborhood graph in order to bound $\mathbf{d}_{P,k}$ in terms of $\mathbf{d}_{M,1}$.

After i points have been added, the inserted points are denoted M_i and the cluster graph is denoted G_i. The neighbors of a point $q \in M_i$ in G_i are denoted $N_i(q)$. We include q itself in $N_i(q)$ for all $q \in M_i$.

If we consider adding a point p, but do not add it because $\mathbf{d}_{P,k}(p) > 2\mathbf{d}_{M,1}(p)$, then this condition will continue to hold as we add more points to M. That is, if we decide not to add p at time i, there will not later come a time when we do want to add it. So, we can safely remove this point from the data structure. We say such a point is marked. In addition to the cluster graph, we store for each point, a list of potentially near points that have been marked. For a point p, we call this list the *nearby marked list of p* and denote it $\texttt{markedpts}(p)$. These lists are used to correctly bound $\mathbf{d}_{P,k}$ for new points considered later in the algorithm (as explained below).

The algorithm works as follows. Start by adding any point to M. All points start unmarked. At step $i+1$, consider adding the farthest unmarked point p to any point in M_i. A heap in the cluster graph makes it easy to find this point. Let $q = \mathrm{NN}_{M_i}(p)$ so $p \in C_q$. Count the number of points in the ball $B = \texttt{ball}(p, 2\mathbf{d}_{M_i,1}(p))$ by iterating over the clusters of q and its neighbors. We also check the nearby marked list of p to count the marked points in B. If there are at least k points in B, then we add p to our kNN sample and continue.

If there are fewer than k points in B, then we mark p and remove it from its cluster. For each of the points

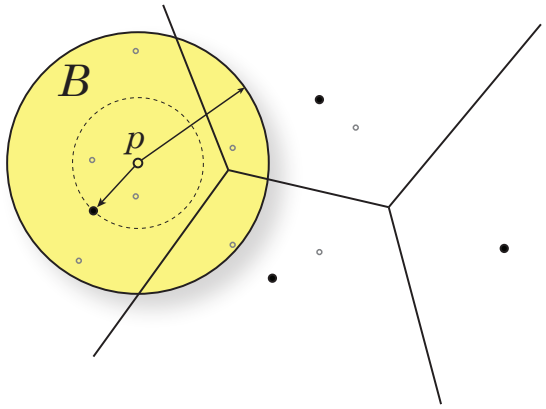


Figure 5: When considering the addition of point p, we count points in $B = \texttt{ball}(p, 2\mathbf{d}_{M,1}(p))$. These are all among the cluster adjacent to the cluster containing p and the list $\texttt{markedpts}(p)$. The clusters are shown as Voronoi cells as a visual aid.

in B that have not yet been added, they may have p nearby when they are considered for insertion later. We add p to the nearby marked list of each point in B. As the following lemma shows, this combination of cluster graph plus nearby marked lists is sufficient to enumerate B.

Lemma 6 *When considering the addition of a point $p \in C_a$ into M_j, the set of points P in $B = \texttt{ball}(p, 2\mathbf{d}_{M_j,1}(p))$ is contained in*

$$\left(\bigcup_{b \sim a} C_b\right) \cup \texttt{markedpts}(p).$$

Proof. By the definition of the cluster graph, the clusters adjacent to a contain all the points that can be within $2\mathbf{d}_{M_j,1}(p)$ of p. However, some points q are removed from the clusters if at some time $i < j$, they were marked because $\mathbf{d}_{P,k}(q) > 2\mathbf{d}_{M_i,1}(q)$. Because the points are considered for insertion in a greedy ordering, we have that if $q \in B$, then

$$\mathbf{d}(p, q) \leq 2\mathbf{d}_{M_j,1}(p) \leq 2\mathbf{d}_{M_i,1}(q).$$

Therefore, q would have been added to $\texttt{markedpts}(p)$ at the time q was marked. □

6 Analysis

In this section we will show that the algorithm of Section 5 computes a kNN sample in $O(kn \log \Delta)$ time for doubling metrics.

The key step of the analysis is to show that each point is touched at most $O(k)$ times before the insertion radius goes down by a constant factor. This is similar to Har-Peled and Mendel's analysis [9] of Clarkson's algorithm except that in our case a point may be considered but not added. The volume packing argument is easily adapted to this case with the loss of a factor of k in the running time.

A point q is *touched* when considering insertion of a point p if we compute $\mathbf{d}(p, q)$. The analysis depends on counting these touches.

Lemma 7 *A point b is touched at most $O(k \log \Delta_k(P))$ times when computing the kNN sample of a doubling metric P.*

Proof. Partition the set of points that touch b into sets A_i where the maximum radius in the cluster graph is in the interval $[2^i, 2^{i+1})$ at the time of a touch from $a \in A_i$. It will suffice to show that there are only $O(k)$ points in each A_i, because at most $O(\log(\Delta_k(P)))$ sets A_i are nonempty. Let S_i be a 2^{i-1} covering of A_i. We have $|A_i| \leq k|S_i|$, because if any point of S_i had more than k points of A_i in its ball of radius 2^{i-1}, one of them would have been inserted and therefore another of them would have been considered for insertion at a time when the radius is less than 2^i, contradicting the assumption that the point is in A_i.

Moreover, for all $a \in A_i$, $\mathbf{d}(a, b) \leq 6 \cdot 2^{i+1}$ by the triangle inequality and the definition of edges in the cluster graph. By the usual packing argument (see Har-Peled and Mendel [9]), this implies that

$$|A_i| \leq k|S_i| \leq k\gamma^{\lceil \log_2 24 \rceil} = O(k).$$

□

Theorem 8 *Let P be a finite metric space with size n, doubling dimension d, and spread Δ. The Greedy kNN sampling algorithm runs time $O(kn \log \Delta)$.*

Proof. The algorithm has several different pieces that must be analyzed separately. In the main loop, there are some heap operations to find the next point to consider for addition. This requires $O(\log n)$ time per point. The local search requires touching all the points in a cluster and its neighbors. This requires $O(k \log \Delta_k(P))$ time per point according to Lemma 7. It also requires touching all the points in the nearby marked list. As each point is added to at most $k - 1$ such lists, there are $O(kn)$ touches of this type. By Corollary 5, updating the cluster graph requires only constant time per vertex. Because $n = O(\Delta^d)$, we have $\log n = O(\log \Delta)$. Using the fact that $\Delta_k \leq \Delta$, the total running time is $O(kn \log \Delta)$. □

7 Software

We have implemented the GreedykNN algorithm and integrated it into the `greedypermutations` python

package. The code can be accessed at `https://github.com/donsheehy/greedypermutation` and the documentation at `https://donsheehy.github.io/greedypermutation/`. The code can be installed with `pip` by running the following from a command line.

```
pip install greedypermutation
```

Here are several examples of the code in use. We start with an example of exponentially-spaced points.

```
from greedypermutation.knnsample import knnsample

P = [Point(1.2**i, 5) for i in range(10, 100)]
S_10 = list(knnsample(P,10))
```

Next, we show uniform points with $k = 10$.

```
P = [Point(i, 5) for i in range(10, 600, 10)]
S_10 = list(knnsample(P,10))
```


The next instance, with $k = 20$ is predictably twice as sparse.

```
P = [Point(i, 5) for i in range(10, 600, 10)]
S_20 = list(knnsample(P,20))
```


8 Conclusion

This paper presented a simple greedy approach to computing kNN samples that have distances between the points bounded within a constant times the distance to k points locally. This balances the desire to have a geometrically nice sample, but also one whose density varies with the underlying distribution.

There are several open problems that remain. First, it is not known whether there are bounds on α and β that are necessary for an (α, β)-kNN sample to exist. From the results in this paper, we know that $(1/5, 2)$-kNN samples exist for any finite metric space, but nothing is known for larger α and smaller β. Also open is whether one can efficiently compute a variation where one uses $\mathbf{d}_{M,k'}$ for some $k' < k$ other than 2. It may be that one can give a tighter approximation, but this question is also open.

Another open question is whether or not there are more efficient algorithms for implementing the simple heuristic of Section 3. In particular, it may be possible to cut out or reduce the factor of k. The reason this seems possible is that it comes from enumerating rather than just counting the points in the metric ball around each point that are considered for addition.

References

[1] G. Biau and L. Devroye. *Lectures on the Nearest Neighbor Method*. Springer Series in the Data Sciences. Springer, 2015.

[2] K. L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

[3] K. L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for 'sb(s)'. Preliminary version presented at ALENEX99, 2003.

[4] K. L. Clarkson. Nearest-neighbor searching and metric space dimensions. In G. Shakhnarovich, T. Darrell, and P. Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

[5] C. D. Cutler and D. A. Dawson. Estimation of dimension for spatially distributed data and related limit theorems. *Journal of Multivariate Analysis*, 28(1):115–148, 1989.

[6] M. Dyer and A. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985.

[7] K. Gardner and D. Sheehy. kth nearest neighbor sampling in the plane. In *Proceedings of the Canadian Conference on Computational Geometry*, 2016.

[8] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[9] 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.

[10] S. Har-Peled and B. Raichel. Net and prune: A linear time algorithm for euclidean distance problems. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, 2013.

[11] M. Izbicki and C. R. Shelton. Faster cover trees. In *Proceedings of the Thirty-Second International Conference on Machine Learning*, 2015.

Appendix

9 How small is the sample?

Ideally, one hopes that the sample M is much smaller than P. If P has n points, then $2n/k$ points is the goal, having 2 points in the sample for k points in the input. Allowing some constant factors we want $|M|$ to be $O(n/k)$. There are clearly cases where this is achieved. For example, points spaced uniformly on a line will have a kNN sample that is uniformly spaced and has size $O(n/k)$.

Unfortunately, there are also simple examples where a kNN sample will be large. One such example is exponentially spaced points on a line, such as $\{10^i \mid i = 1, \ldots, n\}$.

The size decrease is best understood in terms of the measure induced by a kth nearest neighbor density estimate. Let's continue with one dimensional examples. Let $q : [0, 1] \to \mathbb{R}$ be a density function for a measure

$$\mu(B) := \int_{x \in B} q(x)dx.$$

with total mass n, i.e. $\mu([0, 1]) = n$.

The k-th nearest neighbor density estimate constructed from P induces the following approximation to this measure.

$$\mu_{P,k}(B) := \int_{x \in B} \frac{k}{\mathbf{d}_{P,k}(x)}dx.$$

Similarly, we get an estimate from M.

$$\mu_{M,2}(B) := \int_{x \in B} \frac{2}{\mathbf{d}_{M,2}(x)}dx.$$

If M satisfies the kNN sampling lower bound $\alpha\mathbf{d}_{P,k} \leq \mathbf{d}_{M,2}$, then we can relate the total mass of these measures

$$\begin{aligned}\mu_{M,2}([0, 1]) &= \int_0^1 \frac{2}{\mathbf{d}_{M,2}(x)}dx \\ &\leq \frac{2}{k}\int_0^1 \frac{k}{\alpha\mathbf{d}_{P,k}(x)}dx \\ &= \frac{2}{k\alpha}\mu_{p,k}([0, 1])\end{aligned}$$

So, the total mass of the measure $\mu_{M,2}$ is $O(1/k)$ times the total mass of $\mu_{P,k}$. There is nothing special about the line in this example and the same argument holds for other measures induced by densities.

The the total mass of $\mu_{M,2}$ gives an upper bound on the number of points, i.e., $|M| = O(\mu_{M,2}([0, 1]))$. So, the number of points in a kNN sample is $O(1/k)$ times the total mass of $\mu_{P,k}$. This means that if the kNN sample is large, then the kNN density estimate was a bad approximation to the true density. If that is the case, then using kth nearest neighbors may have been a poor choice in the first place.

The moral of this story is that the kNN sample will have size $O(n/k)$ whenever the k-th nearest neighbor density estimate was a good approximation to the underlying measure from which P was sampled.

10 A Python Implementation

The main kNNSample algorithm is described in the prose of the paper. However, the algorithm is sufficiently simple that we can include here the code from the Python implementation.

The cluster graph implementation is straightforward and not shown here. The clusters are iterable and provide `pop` method that returns their farthest point. The cluster graph stores the clusters in a heap, ordered by their radius. It provides a `nbrs_of_nbrs` method that allows one to iterate over all clusters within two hops of a given `cluster`. Because the graph has self loops, this set includes `cluster` and its immediate neighbors. The `knnsample` function is a generator, so it `yield`s points as they are added to the sample rather than returning the final set.

Several small optimizations appear in the official release of the code that have been removed here to show the essentials of the algorithm. That said, the code below is complete and has been tested.

```
from collections import defaultdict
from greedypermutation.clustergraph import ClusterGraph

def knnsample(M, k, seed = None):
    G = ClusterGraph(M, nbrconstant = 2, moveconstant = 1)
    markedpts = defaultdict(set)

    # Yield the first point.
    yield G.heap.findmax().center

    for i in range(1, len(M)):
        cluster = G.heap.findmax()
        point = cluster.pop()
        G.heap.changepriority(cluster)
        radius = 2 * point.dist(cluster.center)

        nearbypts = {q for nbr in G.nbrs(cluster)
                       for q in nbr
                       if q.dist(point) <= radius
                    }

        nearbymarkedpts = {q for q in markedpts[point]
                             if q.dist(point) <= radius
                           }

        if len(nearbypts) + len(nearbymarkedpts) < k:
            for p in nearbypts:
                markedpts[p].add(point)
        else:
            G.addcluster(point, cluster)
            yield point
```

Social Distancing is Good for Points too!

Alejandro Flores-Velazco*

Abstract

The *nearest-neighbor rule* is a well-known classification technique that, given a training set P of labeled points, classifies any unlabeled query point with the label of its closest point in P. The *nearest-neighbor condensation* problem aims to reduce the training set without harming the accuracy of the nearest-neighbor rule.

FCNN is the most popular algorithm for condensation. It is heuristic in nature, and theoretical results for it are scarce. In this paper, we settle the question of whether reasonable upper-bounds can be proven for the size of the subset selected by FCNN. First, we show that the algorithm can behave poorly when points are too close to each other, forcing it to select many more points than necessary. We then successfully modify the algorithm to avoid such cases, thus imposing that selected points should "keep some distance". This modification is sufficient to prove useful upper-bounds, along with approximation guarantees for the algorithm.

1 Introduction

In the context of non-parametric classification, a *training set* P consists of n points in a metric space $(\mathcal{X}, \mathsf{d})$, with domain $\mathcal{X}$ and distance function $\mathsf{d} : \mathcal{X}^2 \to \mathbb{R}^+$. Additionally, P is partitioned into a finite set of *classes* by associating each point $p \in P$ with a *label* $l(p)$, indicating the class to which it belongs. Given an *unlabeled* query point $q \in \mathcal{X}$, the goal of a *classifier* is to predict q's label using the training set P.

The *nearest-neighbor rule* is among the best-known classification techniques [5]. It assigns a query point the label of its closest point in P, according to the metric d. The nearest-neighbor rule exhibits good classification accuracy both experimentally and theoretically [3, 4, 14], but it is often criticized due to its high space and time complexities. Clearly, the training set P must be stored to answer nearest-neighbor queries, and the time required for such queries depends to a large degree on the size and dimensionality of the data. These drawbacks inspire the question of whether it is possible replace P with a significantly smaller subset, without significantly reducing the classification accuracy under the nearest-neighbor rule. This problem has been widely studied, and it is often called *nearest-neighbor condensation* [8, 9, 13, 15].

*University of Maryland, College Park, afloresv@cs.umd.edu

Related work. A subset $R \subseteq P$ is said to be *consistent* if and only if for every $p \in P$ its nearest-neighbor in R is of the same class as p. Intuitively, R is consistent [9] if and only if all points of P are correctly classified using the nearest-neighbor rule over R. Formally, the problem of nearest-neighbor condensation consists of finding an ideally small consistent subset of P.

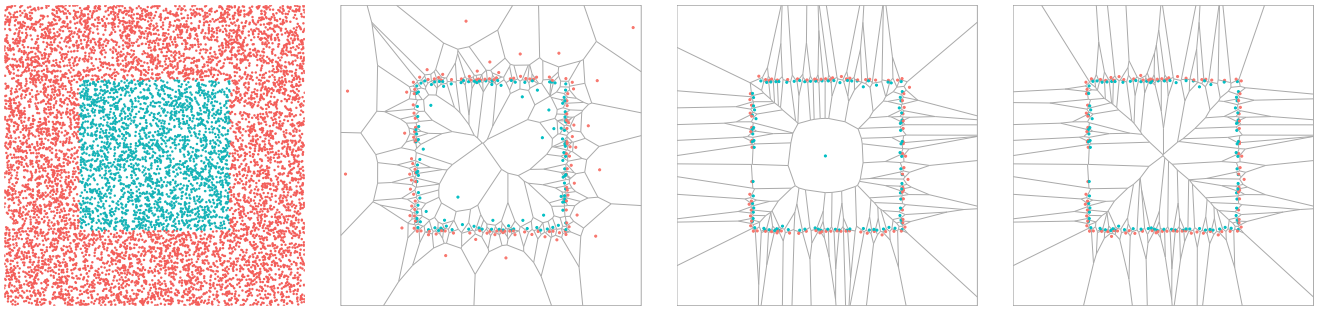
It is known that the problem of computing consistent subsets of minimum cardinality is NP-hard [12, 17, 18]. However, there exists an algorithm called NET [8] that computes a tight approximation of the minimum cardinality consistent subset. Yet, this algorithm is not practical, and it is often outperformed on real-world training sets —with respect to both their runtime and size of the selected subsets— by simple heuristics for condensation.

Most algorithmic research for this problem has focused on heuristics; for comprehensive surveys, see [11, 15, 16]. Out of the many heuristics proposed for this problem, FCNN [1] stands out due to its quadratic worst-case time complexity, and most importantly, its observed efficiency when applied to real-world training sets. Alternatives include CNN [9], MSS [2], RSS [6], and VSS [6]. These algorithms also run in quadratic time, except for CNN, which has cubic runtime, and was the first algorithm proposed for condensation. See Figure 1 for an illustrative comparison between these heuristics.

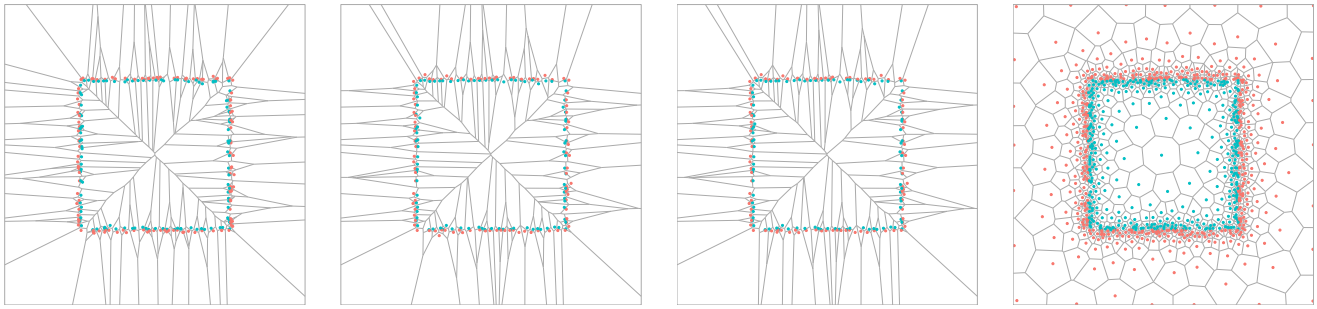
While such heuristics have been extensively studied experimentally [7], theoretical results are scarce. Only recently in CCCG'19 [6], we have shown that the size of the subset selected by MSS cannot be bounded. On the other hand, we proved that the size of the subset selected by both RSS and VSS can be upper-bounded. However, until now, it remained open whether similar results could be achieved for FCNN.

Contributions. In this paper, we settle the question of whether the size of the subsets selected by FCNN can be upper-bounded. Our results are summarized as follows:

- There exist training sets for which the subset selected by FCNN is unbounded, particularly, when compared to the selection of other algorithms (*e.g.*, RSS).
- We propose a modification of FCNN, namely SFCNN, for which we prove the following results:
  - The size of the subset selected by SFCNN has an upper-bound, similar to the one known for RSS.
  - SFCNN computes a tight approximation of the minimum cardinality consistent subset of P.



(a) Training set (10^4 pts) (b) CNN (253 pts) (c) FCNN (185 pts) (d) SFCNN (190 pts)



(e) MSS (234 pts) (f) RSS (192 pts) (g) VSS (193 pts) (h) NET (841 pts)

Figure 1: An illustrative example of the subsets selected by different condensation algorithms from an initial training set P in $\mathbb{R}^2$ of 10^4 points. The list includes well-known algorithms like CNN, FCNN, MSS, RSS, VSS, and NET. We propose SFCNN as a simple modification of FCNN that can be successfully upper-bounded.

Preliminaries. Given any point $q \in \mathcal{X}$ in the metric space, its nearest-neighbor, denoted by $\text{nn}(q)$, is the closest point of P according the the distance function d. Given a point $p \in P$, any other point of P whose label differs from p's is called an *enemy* of p. The closest such point is called p's *nearest-enemy*, denoted by $\text{ne}(p)$.

Clearly, the size of a condensed subset should depend on the spatial characteristics of the classes in the training set. For example, a consistent subset for two spatially well separated clusters should be smaller than the subset for one with two classes that have a high degree of overlap. To model this intrinsic complexity, define κ to be the number of nearest-enemy points of P, *i.e.*, the cardinality of set $\{\text{ne}(p) \mid p \in P\}$.

This has been previously used [6] to prove useful upper-bounds for RSS, and to show negative results for MSS. In particular, it has been shown that RSS selects $\mathcal{O}(\kappa\,(3/\pi)^{d-1})$ points in d-dimensional Euclidean space, while MSS's selection cannot be bounded in terms of κ.

2 Nearby Points are Problematic

Now consider the FCNN algorithm [1]. It follows an iterative incremental approach to build a consistent subset of P (see Algorithm 1 for a formal description). While not immediately evident, FCNN it runs in $\mathcal{O}(nm)$ worst-case time, where m is the final size of the selected subset.

Algorithm 1: FCNN

Input: Initial training set P
Output: Consistent subset $R \subseteq P$

```
R ← φ
S ← centroids(P)
while S ≠ φ do
    R ← R ∪ S
    S ← φ
    foreach p ∈ R do
        S ← S ∪ {rep(p, voren(p, R, P))}
return R
```

The algorithm begins by selecting one point per class, in particular, the centroid of each class[1]. Then, it begins the iterative process, selecting other points until the subset is consistent. During each iteration, the algorithm identifies all points of P that are misclassified with respect to the current subset, and adds some of these points to the subset. Formally, for every point p already in the subset, FCNN selects one *representative* among non-selected points, whose nearest-neighbor is p and that belong to a different class than

[1]For each class, its centroid is defined as the closest point of P to the geometrical center of all points of this class.

p. That is, the representative is selected from the set $\text{voren}(p, R, P) = \{q \in P \mid \text{nn}(q, R) = p \wedge l(q) \neq l(p)\}$. Usually, the representative chosen is the one closest to p, although different approaches can be applied.

However, there is an issue with this algorithm. During any given iteration, nothing prevents the representatives of two neighboring points in FCNN to be arbitrarily close to each other. This observation can be exploited to obtain the following result:

Theorem 1 *There exists a training set $P \subset \mathbb{R}^d$ in Euclidean space, with constant number of classes, for which* FCNN *selects $\Omega(\kappa/\xi)$ points, for any $0 < \xi < 1$.*

The remaining of this section addresses the proof of this theorem, by carefully constructing a training set P in $\mathbb{R}^3$ that exhibits the undesirable behavior in the selection process of the FCNN algorithm.

Without loss of generality, let $\xi = 1/2^t$ for some value $t > 3$, we construct a training set $P \subset \mathbb{R}^3$ with a constant number of classes and the number of nearest-enemy points κ equal to $\mathcal{O}(1/\xi)$, for which FCNN is forced to select $\mathcal{O}(1/\xi^2)$ points. As mentioned above, the key downside of the algorithm occurs when points are added to the subset in the same iteration, as they can be arbitrarily close to each other. We exploit this behavior to force the algorithm to select $\mathcal{O}(1/\xi)$ such points on each iteration.

Intuitively, the training set P consists of several layers of points arranged parallel to the xy-plane, and stacked on top of each other around the z-axis (see Figure 2). Each layer is a disk-like arrangement, formed by a center point and points at distance 1 from this center. Define the *backbone points* of P to be the center points $c_i = 2i\vec{v}_z$ for $i \geq 0$. We now describe the different arrangements of points as follows (see Figure 2):

$$\begin{aligned}
\mathcal{B} &= c_0 \cup \{y_j = c_0 + \vec{v}_x R_z(j\pi/4)\}_{j=1}^{8} \\
\mathcal{M}_i &= \{c_{2i}, c_{2i+1}, m_i = (c_{2i} + c_{2i+1})/2\} \\
&\quad \cup \left\{r_{ij} = c_{2i} + \vec{v}_x R_z\left(j\pi/2^{1+i}\right)\right\}_{j=1}^{2^{2+i}} \\
&\quad \cup \left\{b_{ij} = c_{2i+1} + \vec{v}_x R_z\left(j\pi/2^{1+i}\right)\right\}_{j=1}^{2^{2+i}} \\
&\quad \cup \left\{w_{ij} = c_{2i+1} + \vec{v}_x R_z\left((j+1/2)\pi/2^{1+i} - \xi^2\right)\right\}_{j=1}^{2^{2+i}} \\
\mathcal{R}_i &= \{c_{2i}, c_{2i+1}\} \\
&\quad \cup \{r_{ij} = c_{2i} + \vec{v}_x R_z\left(2j\pi\xi\right)\}_{j=1}^{1/\xi} \\
&\quad \cup \{b_{ij} = c_{2i+1} + \vec{v}_x R_z\left(2j\pi\xi\right)\}_{j=1}^{1/\xi}
\end{aligned}$$

These points belong to one of 11 classes, defined by the set $\{1, \ldots, 8, \text{red}, \text{blue}, \text{white}\}$. Then, we define the labeling function l as follows: $l(c_i)$ is red when i is even and blue when i is odd, $l(m_i)$ is white, $l(y_j)$ is the j-th class, $l(r_{ij})$ is red, $l(b_{ij})$ is blue, and $l(w_{ij})$ is white.

Base arrangement ($\mathcal{B}$). Consists of one single layer of points, with one red center point c_0 and 8 points y_j in the circumference of the unit disk (parallel to the xy-plane), each labeled with a unique class j (see Figure 2d).

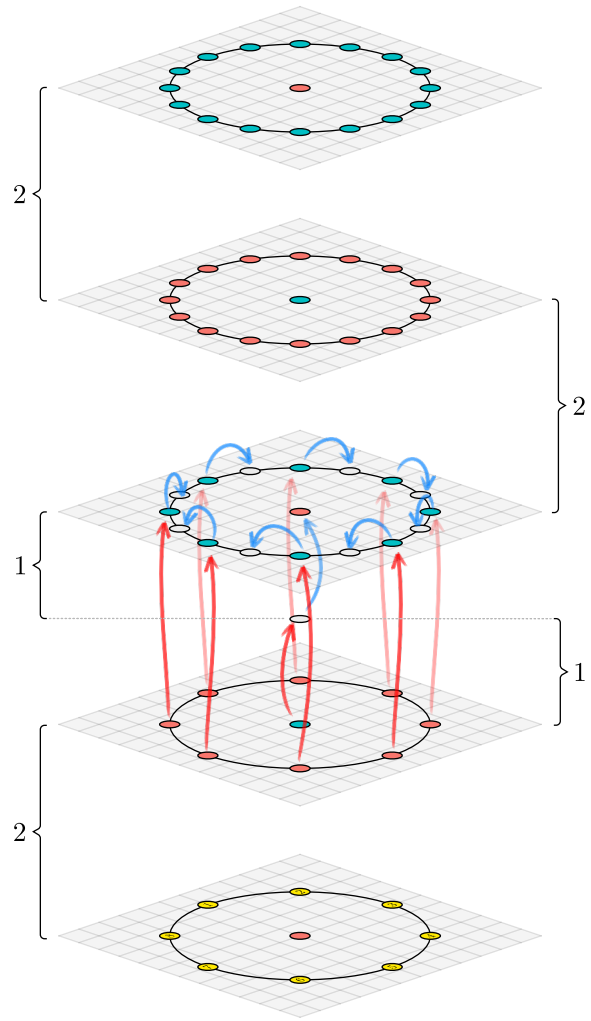
The goal of this arrangement is that each of these points is the centroid of its corresponding class. The centroids of the blue and white classes can be fixed to be far enough, so we won't consider them for now. Hence, the first iteration of FCNN will add all the points of $\mathcal{B}$. In the next iteration, each of these points will select a representative in the arrangement above. Clearly, the size of $\mathcal{B}$ is 9, and it contributes with 8 nearest-enemy points in total.

Multiplier arrangement ($\mathcal{M}_i$). Our final goal is to have $\mathcal{O}(1/\xi)$ arbitrarily close points selecting representatives on a single iteration;. Initially, we only have 9, the ones in the base arrangement. While this could be simply achieved with $\mathcal{O}(1/\xi)$ points in $\mathcal{B}$ each with a unique class, we want to use a constant number of classes. Instead, we use each multiplier arrangement to double the number of representatives selected.

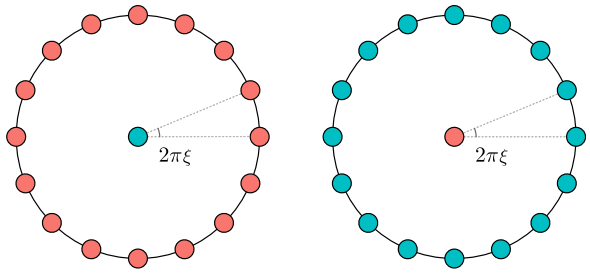
$\mathcal{M}_i$ consists of (1) a layer with a blue center c_{2i} and 2^{2+i} red points r_{ij} around the unit disk's circumference, (2) a layer with a red center c_{2i+1} and 2^{3+i} blue b_{ij} and white w_{ij} points around the unit disk's circumference, and (3) a white center point m_i in the middle between the red and blue center points (see Figure 2c). Suppose at iteration $3i-1$ all the points r_{ij} and c_{2i} of the first layer are added as representatives of the previous arrangement, which is given for $\mathcal{M}_1$ from the selection of $\mathcal{B}$. Then, during iteration $3i$ each r_{ij} adds the point b_{ij} right above, while c_{2i} adds point m_i (see the red arrows in Figure 2a). Finally, during iteration $3i+1$, m_i adds c_{2i+1}, and each b_{ij} adds point w_{ij} as its the closest point inside the voronoi cell of b_{ij} (see the blue arrows in Figure 2a). Now, with all the points of this layer added, each continues to select points in the following arrangement (either $\mathcal{M}_{i+1}$ or $\mathcal{R}_{i+1}$).

The size of each $\mathcal{M}_i$ is $3(1 + 2^{2+i}) = \mathcal{O}(2^{3+i})$, and contributes with $3 + 2(2^{2+i}) = \mathcal{O}(2^{3+i})$ to the total number of nearest-enemy points. Thus, we stack $\mathcal{M}_i$ arrangements for $i \in [1, \ldots, t-3]$, such that the last of these selects $1/\xi = 2^t$ points.

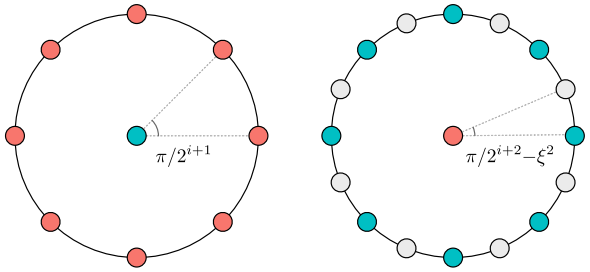
Repetitive arrangement ($\mathcal{R}_i$). Once the algorithm reaches the last multiplier layer $\mathcal{M}_{t-3}$, it will select $1/\xi$ points during the following iteration. The repetitive arrangement allows us to continue selecting these many points on every iteration, while only increasing the number of nearest-enemy points by a constant. This arrangement consists of (1) a first layer with a blue center c_{2i} surrounded by $1/\xi$ red points r_{ij} around the unit disk circumference, and (2) a second layer with red center c_{2i+1} and blue points b_{ij} in the circumference (see



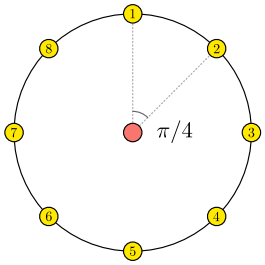
(a) Entire arrangement of points, by stacking the different arrangements along the z-axis. The arrows illustrate the selection process of FCNN on a multiplicative arrangement $\mathcal{M}_i$.



(b) A repetitive arrangement $\mathcal{R}_i$. This is used to maintain the number of selected representatives to be $\mathcal{O}(1/\xi)$ during each remaining iteration of the algorithm.



(c) A multiplier arrangement $\mathcal{M}_i$. This forces FCNN to double the number of selected representatives around the circumference after two iterations.



(d) Base arrangement $\mathcal{B}$. Each point in the circumference belongs to one unique class $\{1,\dots,8\}$, here colored in yellow and numbered for clarity.

Figure 2: Example of a training set $P \subset \mathbb{R}^3$ for which FCNN selects $\Omega(\kappa/\xi)$ points.

Figure 2b). Once the first layer is added all in a single iteration, during the following iteration c_{2i} adds c_{2i+1}, and each r_{ij} adds b_{ij}.

The size of each $\mathcal{R}_i$ is $2(1+1/\xi) = \mathcal{O}(1/\xi)$, and it contributes with 4 points to the total number of nearest-enemy points. Now, we stack $\mathcal{O}(1/\xi)$ such arrangements $\mathcal{R}_i$ for $i \in [t-2,\dots,1/\xi]$, such that we obtain the desired ratio between selected points and number of nearest-enemy points of the training set.

The training set. After defining all the necessary point arrangements, and recalling that $t = \log 1/\xi$, we put these arrangements together to define the training set P as follows:

$$P = \mathcal{B} \bigcup_{i=1}^{t-3} \mathcal{M}_i \bigcup_{i=t-2}^{1/\xi} \mathcal{R}_i \cup \mathcal{F}$$

where $\mathcal{F}$ is an additional set of points to fix the centroids of P. These extra points are located far enough from the remaining points of P, and are carefully placed such that the centroids of P are all the points of $\mathcal{B}$, plus a blue and white point from $\mathcal{F}$. Additionally, all the points of $\mathcal{F}$ should be closer to its corresponding class centroid than to any enemy centroid, and they should increase the number of nearest-enemy points by a constant. This can be done with a bounded number of extra points.

All together, by adding up the corresponding terms, the ratio between the size of FCNN and κ (the number

of nearest-enemy points of P) is $\mathcal{O}(1/\xi)$. Therefore, on this training set, FCNN selects $\mathcal{O}(\kappa/\xi)$ points.

3 Keeping Distance: One by One

Evidently, adding points in batch on every iteration of the algorithm prevents FCNN to have provable guarantees on the size of its selected subset, just as RSS provides. However, this design choice is not key for any of the features of the algorithm.

Therefore, we propose to modify FCNN such that only *one* single representative is added to the subset on each iteration. We call this new algorithm SFCNN or *Single* FCNN. Basically, the only difference between the original FCNN and SFCNN is on line 4 of Algorithm 1, where R is updated by selecting one single point from the set of representatives S, as follows:

$$R \leftarrow R \cup \{\text{Choose one point of } S\}$$

While extremely simple, this change in the selection process of SFCNN allows us to successfully analyze the size of its selected subset in terms of κ, and even prove that it approximates the consistent subset of minimum cardinality.

Size Upper-Bound. To this end, we first need to introduce some terminology. Through a suitable uniform scaling, we may assume that the *diameter* of P (that is, the maximum distance between any two points in the training set) is 1. The *spread* of P, denoted as Δ, is the ratio between the largest and smallest distances in P. Define the *margin* of P, denoted γ, to be the smallest nearest-enemy distance in P. Clearly, $1/\gamma \leq \Delta$.

The metric space $(\mathcal{X}, \mathsf{d})$ is said to be *doubling* [10] if there exist some bounded value λ such that any metric ball of radius r can be covered with at most λ metric balls of radius $r/2$. Its *doubling dimension* is the base-2 logarithm of λ, denoted as $\text{ddim}(\mathcal{X}) = \log \lambda$. Throughout, we assume that $\text{ddim}(\mathcal{X})$ is a constant, which means that multiplicative factors depending on $\text{ddim}(\mathcal{X})$ may be hidden in our asymptotic notation. Many natural metric spaces of interest are doubling, including d-dimensional Euclidean space whose doubling dimension is $\Theta(d)$. It is well know that for any subset $R \subseteq \mathcal{X}$ with some spread Δ_R, the size of R is bounded by $|R| \leq \lceil \Delta_R \rceil^{\text{ddim}(\mathcal{X})+1}$.

Theorem 2 SFCNN *selects a subset of size:*

$$\mathcal{O}\left(\kappa \log \frac{1}{\gamma}\ 4^{\text{ddim}(\mathcal{X})+1}\right)$$

Proof. This follows by a charging argument on each nearest-enemy point in the training set. Consider one such point $p \in \{\text{ne}(r) \mid r \in P\}$ and a value $\sigma \in [\gamma, 1]$. We define $R_{p,\sigma}$ to be the subset of points from SFCNN whose nearest-enemy is p, and whose distance to p is between σ and 2σ. That is, $R_{p,\sigma} = \{r \in R \mid \text{ne}(r) = p \wedge \mathsf{d}(r,p) \in [\sigma, 2\sigma)\}$. These subsets define a partitioning of R when considering all nearest-enemy points of P, and values of $\sigma = \gamma\, 2^i$ for $i = \{0, 1, 2, \ldots, \lceil \log \frac{1}{\gamma} \rceil\}$.

Consider any two points $a, b \in R_{p,\sigma}$ in these subsets. Assume *w.l.o.g.* that point a was selected by the algorithm before point b (*i.e.*, in a prior iteration). We show that $\mathsf{d}(a,b) \geq \sigma$. By contradiction, assume that $\mathsf{d}(a,b) < \sigma$, which immediately implies that a and b belong to the same class. Moreover, recalling that b's nearest-enemy in P is p, at distance $\mathsf{d}(b,p) \geq \sigma$, this implies that b is closer to a than to any enemy in R. Therefore, by the definition of the *voren* function, b could never be selected by SFCNN, which is a contradiction.

This proves that $\mathsf{d}(a,b) \geq \sigma$. Additionally, we know that $\mathsf{d}(a,b) \leq \mathsf{d}(a,p) + \mathsf{d}(p,b) \leq 4\sigma$. Thus, using a simple packing argument with the known properties of doubling spaces, we have that $|R'_{p,\sigma}| \leq 4^{\text{ddim}(\mathcal{X})+1}$.

Altogether, by counting over all the $R_{p,\sigma}$ sets for every nearest-enemy in the training set and values of σ, the size of R is upper-bounded by $|R| \leq \kappa \lceil \log 1/\gamma \rceil\, 4^{\text{ddim}(\mathcal{X})+1}$. This completes the proof. □

An Approximation Algorithm. Denote Min-CS as the problem of computing a minimum cardinality consistent subset of P. This problem is known to be NP-hard [12, 17,18], even to approximate [8] in polynomial time within a factor of $2^{(\text{ddim}(\mathcal{X}) \log (1/\gamma))^{1-o(1)}}$.

As previously mentioned, the NET algorithm [8] computes a tight approximation for the Min-CS problem. The algorithm is rather simple: it just computes a γ-net of P, where γ is the margin (the smallest nearest-enemy distance in P). This clearly results in a consistent subset of P, whose size is at most $\lceil 1/\gamma \rceil^{\text{ddim}(\mathcal{X})+1}$. A similar result can be proven for SFCNN.

Theorem 3 SFCNN *computes a tight approximation for the* Min-CS *problem.*

Proof. This follows from a direct comparison to the resulting subset of the NET algorithm. For any point $p \in$ NET, let B_p be the set of points of P "covered" by p, that is, whose distance to p is at most γ. By the covering property of nets, this defines a partition on P when considering every point p selected by NET.

Let's analyze the size of $B_p \cap R$, that is, for any given B_p how many points could have been selected by the SFCNN algorithm. Let $a, b \in B_p \cap R$ be two such points, where without loss of generality, point a was selected in an iteration before b. Both a and b must belong to the same class as p, as their distance to p is at most γ, which is the smallest nearest-enemy distance in P. Moreover, by the definition of the *voren* function, it is easy to show that $\mathsf{d}(a,b) \geq \gamma$. By a simple packing argument in doubling metrics, the size of any $B_p \cap R$ is at most $2^{\text{ddim}(\mathcal{X})+1}$. All together, we

have that the size of the subset selected by SFCNN is $2^{\mathrm{ddim}(\mathcal{X})+1}\,|\mathrm{NET}| = \mathcal{O}\left((1/\gamma)^{\mathrm{ddim}(\mathcal{X})+1}\right)$. □

4 Experimental Results

The importance of FCNN relies on its performance in practice, despite the lack of theoretical guarantees. A natural question is whether the simple change we proposed on the algorithm, negatively affects its performance in real-world training sets.

Thus, to get a clearer impression of the relevance of these results in practice, we performed experimental trials on several training sets, both synthetically generated and widely used benchmarks. First, we consider 21 training sets from the UCI *Machine Learning Repository*[2] which are commonly used in the literature to evaluate condensation algorithms [7]. These consist of a number of points ranging from 150 to 58000, in d-dimensional Euclidean space with d between 2 and 64, and 2 to 26 classes. We also generated some synthetic training sets, containing 10^5 uniformly distributed points, in 2 to 3 dimensions, and 3 classes. All training sets used in these experimental trials are summarized in Table 1. The implementation of the algorithms, training sets used, and raw results, are publicly available[3].

We test 7 different condensation algorithms, namely FCNN, SFCNN, RSS, VSS, MSS, CNN and NET. To compare their results, we consider their runtime and the size of the selected subset. Clearly, these values might differ greatly on training sets whose size are too distinct. Therefore, before comparing the raw results, these are normalized. The runtime of an algorithm for a given training set is normalized by dividing it by n, the size of the training set. The size of the selected subset is normalized by dividing it by κ, the number of nearest-enemy points in the training set, which characterizes the complexity of the boundaries between classes.

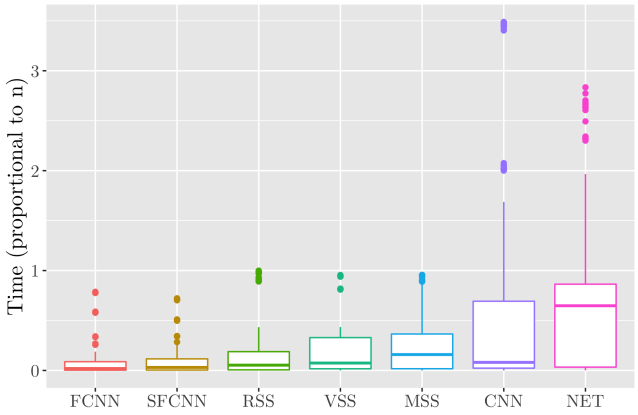


Figure 3: Running time.

[2] https://archive.ics.uci.edu/ml/index.php
[3] https://github.com/afloresv/nnc/

Training set	n	d	c	κ (%)
banana	5300	2	2	811 (15.30%)
cleveland	297	13	5	125 (42.09%)
glass	214	9	6	87 (40.65%)
iris	150	4	3	20 (13.33%)
iris2d	150	2	3	13 (8.67%)
letter	20000	16	26	6100 (30.50%)
magic	19020	10	2	5191 (27.29%)
monk	432	6	2	300 (69.44%)
optdigits	5620	64	10	1245 (22.15%)
pageblocks	5472	10	5	429 (7.84%)
penbased	10992	16	10	1352 (12.30%)
pima	768	8	2	293 (38.15%)
ring	7400	20	2	2369 (32.01%)
satimage	6435	36	6	1167 (18.14%)
segmentation	2100	19	7	398 (18.95%)
shuttle	58000	9	7	920 (1.59%)
thyroid	7200	21	3	779 (10.82%)
twonorm	7400	20	2	1298 (17.54%)
wdbc	569	30	2	123 (21.62%)
wine	178	13	3	37 (20.79%)
wisconsin	683	9	2	35 (5.12%)
v-100000-2-3-15	100000	2	3	1909 (1.90%)
v-100000-2-3-5	100000	2	3	788 (0.78%)
v-100000-3-3-15	100000	3	3	7043 (7.04%)
v-100000-3-3-5	100000	3	3	3738 (3.73%)
v-100000-4-3-15	100000	4	3	13027 (13.02%)
v-100000-4-3-5	100000	4	3	10826 (10.82%)
v-100000-5-3-15	100000	5	3	22255 (22.25%)
v-100000-5-3-5	100000	5	3	17705 (17.70%)

Table 1: Training sets used to evaluate the performance of condensation algorithms. Indicates the number of points n, dimensions d (Euclidean space), classes c, nearest-enemy points κ (also in percentage *w.r.t.* n).

Figures 3 and 4 summarize the experimental results. Evidently, the performance of SFCNN is equivalent to the original FCNN algorithm, both in terms of runtime and the size of their selected subsets, showing that the proposed modification does not affect the behavior of the algorithm in real-world training sets. Both FCNN and SFCNN outperform other condensation algorithms in terms of runtime, while their subset size is comparable in all cases, with the exception of the NET algorithm.

References

[1] F. Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

[2] R. Barandela, F. J. Ferri, and J. S. Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(06):787–806, 2005.

[3] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 1967.

[4] L. Devroye. On the inequality of cover and hart in nearest neighbor discrimination. *Pattern Analysis and*

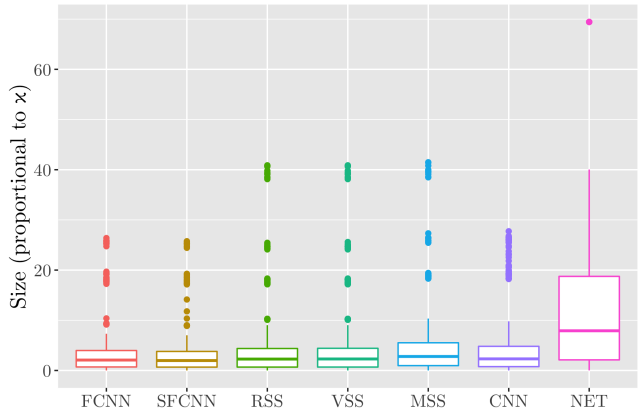


Figure 4: Size of the selected subsets.

Machine Intelligence, IEEE Transactions on, (1):75–78, 1981.

[5] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, Jan. 1951.

[6] A. Flores-Velazco and D. M. Mount. Guarantees on nearest-neighbor condensation heuristics. In Z. Friggstad and J. D. Carufel, editors, *Proceedings of the 31st Canadian Conference on Computational Geometry, CCCG 2019, August 8-10, 2019, University of Alberta, Edmonton, Alberta, Canada*, pages 87–93, 2019.

[7] S. Garcia, J. Derrac, J. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE TPAMI*, 2012.

[8] L.-A. Gottlieb, A. Kontorovich, and P. Nisnevitch. Near-optimal sample compression for nearest neighbors. In *Advances in Neural Information Processing Systems*, 2014.

[9] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.*, 1968.

[10] J. Heinonen. *Lectures on analysis on metric spaces*. Springer Science & Business Media, 2012.

[11] N. Jankowski and M. Grochowski. Comparison of instances selection algorithms I. Algorithms survey. In *Artificial Intelligence and Soft Computing-ICAISC*. 2004.

[12] K. Khodamoradi, R. Krishnamurti, and B. Roy. Consistent subset problem with two labels. In *Conference on Algorithms and Discrete Applied Mathematics*, 2018.

[13] G. L. Ritter, H. B. Woodruff, S. R. Lowry, and T. L. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 1975.

[14] C. J. Stone. Consistent nonparametric regression. *The annals of statistics*, pages 595–620, 1977.

[15] G. Toussaint. Open problems in geometric methods for instance-based learning. In *JCDCG*, volume 2866 of *Lecture Notes in Computer Science*. Springer, 2002.

[16] G. Toussaint. Proximity graphs for nearest neighbor decision rules: Recent progress. In *Progress", Proceedings of the 34th Symposium on the INTERFACE*, pages 17–20, 2002.

[17] G. Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SoCG, pages 224–233, New York, NY, USA, 1991. ACM.

[18] A. V. Zukhba. NP-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recog. Image Anal.*, 20(4):484–494, 2010.

Author Index

Aghamolaei, Sepideh 165
Aldana-Galván, Israel 253
Alegría-Galicia, Carlos 253
Allen, Addison 28
Amano, Kazuyuki 68
Amenta, Nina 209
Arluck, Chloe 223
Arseneva, Elena 54

Bahoo, Yeganeh 54
Belton, Robin Lynne 18
Bercea, Ioana 129
Biedl, Therese 230
Biniaz, Ahmad 49, 54, 230, 346
Bremner, David 272

Cano, Pilar 54
Cano, Rafael 265
Cavanna, Nicholas 78
Chambers, Erin 353
Chanchary, Farah 54

Daescu, Ovidiu 296
Damian, Mirela 189
de Rezende, Pedro 265
de Souza, Cid 265
Duggirala, Parasara 340

Eppstein, David 98

Fasy, Brittany Terese 18
Feder, Tomas 304
Flatland, Robin 189
Fox, Kyle 296

Gabor, Jonathan 42
Garcia, Luis 217
Garciia, Alfredo 49
Ghodsi, Mohammad 165
Gholami Rudi, Ali 334
Goodrich, Michael 98
Goodrich, Michael T. 2

Gutierrez, Andres 217

Hamedmohseni, Bardia 311
Hashemi, Seyed Naser 72
He, Xiaozhou 85
Hell, Pavol 304
Horiyama, Takashi 360
Hou, Kaiying 114
Hsiang, Tien-Ruey 247

Iacono, John 54
Imanparast, Mahdi 72
Irvine, Veronika 230

Jain, Kshitij 54
Janardan, Ravi 282
Johnson, Matthew P. 259
Johnson, Timothy 2
Jorgensen, Jordan 98
Ju, Tao 353

Katz, Matthew 1
Keikha, Vahideh 142
Kim, Woojin 180
Kindermann, Philipp 230
Kisielius, Oliver 78

Lai, Wei-Yu 247
Letscher, David 353
Li, Mao 353
Li, Yuan 282
Liaw, Christopher 172
Liu, Paul 172
Liu, Zhihui 85
Lubiw, Anna 54
Lynch, Jayson 114
Löffler, Maarten 142

Maheshwari, Anil 288, 346
Marin-Nevárez, Nestaly 253
Mastakas, Konstantinos 318
Masterjohn, Joseph 91
Memoli, Facundo 180
Mertz, Rostik 18
Micka, Samuel 18
Milenkovic, Victor 91, 223

Millman, David L. 18
Miyasaka, Masahiro 360
Mohades, Ali 72, 142
Mondal, Debajyoti 54, 311
Mutzel, Petra 11

Nakano, Shin-Ichi 68
Naredla, Anurag Murty 230
Nouri, Arash 288

O'Rourke, Joseph 149, 328
Oettershagen, Lutz 11

Packer, Daniel 35
Pedersen, Logan 158

Rahmati, Zahed 142, 311
Reiss, Robert 172
Rogers, Emmely 328
Rojas, Carlos 209
Ruiz, Isaac 217

Sack, Jörg-Rüdiger 288
Sacks, Elisha 91, 223
Salinas, Daniel 18
Sasaki, Riku 360
Schenfisch, Anna 18
Schupbach, Jordan 18
Shahsavarifar, Rasoul 272
Sheehy, Don 78, 340
Sheikhan, Khadijeh 54
Smid, Michiel 346
Solís-Villarreal, Erick 253
Su, Bing 85
Subi, Carlos 304

Teo, Ka Yaw 296
Topp, Christopher 353
Torres, Manuel 98
Toth, Csaba D. 54
Turcotte, Alexi 230

Uno, Takeaki 61
Urrutia, Jorge 253

van Kreveld, Marc 326

Velarde, Carlos 253

Wang, Haitao 158
Wasa, Kunihiro 61
Wenk, Carola 155
White, Sophia 35
Wilhelm, Martin 367
Williams, Aaron 28, 35, 42
Williams, Lucia 18
Winslow, Andrew 217

Xu, Yinfeng 85
Xue, Jie 282

Yamanaka, Katsuhisa 61
Yan, Yajie 353

Zheng, Feifeng 85
Zhu, Binhai 85

Álvarez-Rebollar, José Luis 253