Proceedings of the 36th Canadian Conference on Computational Geometry (CCCG 2024) Brock University, St. Catharines, Canada

Rahnuma Islam Nishat, Brock University

July 17 - 19, 2024

Copyright information:
Compilation copyright @ 2024 Rahnuma Islam Nishat.
Copyright of individual papers retained by authors.

Credits:
Compilation of the papers: Bastien Rivier
Logo design: Justin Steepe
Proceedings formatting: Denis Pankratov

Welcome Note from the Conference Chair

This volume contains the papers presented at the *CCCG 2024: 36th Canadian Conference on Computational Geometry (CCCG 2024)*, held on July 17 - 19, 2024 at Brock University, St. Catharines, Canada.

For the first time in CCCG, the review process was double blind. There were 49 paper submitted to CCCG 2024; none of the submissions was withdrawn during the review process. After careful reviews, the program committee decided to accept 39 papers.

This year, we introduced the **Best Dissertation Award**. There were 4 doctoral dissertation submitted in this category. the steering committee reviewed the submissions, and selected the best dissertation which was presented by the awardee at the conference.

We thank the authors of all submitted papers and dissertations, and the presenters at the conference. We especially thank the invited speakers: Dr. Vašek Chvátal (Paul Erdős Memorial Talk), Dr. Anna Lubiw (Godfried Toussaint Memorial Talk), and Dr. Mark Keil (Ferran Hurtado Memmorial Talk) for their wonderful talks.

We acknowledge the generous support from our sponsors: Brock University, Concordia University, and Toronto Metropolitan University. In addition, we are grateful for the support and assistance provided by the Department of Computer Science, the Faculty of Mathematics & Science, and the conference services at Brock University. The organizing committee members and the student volunteers worked hard to make the conference a success, we express our gratitude to them.

Rahnuma Islam Nishat

Chair of the Program Committee and the Organizing Committee, CCCG 2024

Sponsors


Brock
University


UNIVERSITÉ
Concordia
UNIVERSITY


Toronto
Metropolitan
University

Program Committee

Oswin Aichholzer	Graz University of Technology
Hugo Akitaya	University of Massachusetts Lowell
Binay Bhattacharya	Simon Fraser University
Therese Biedl	University of Waterloo
Prosenjit Bose	Carleton University
David Bremner	University of New Brunswick
Hsien-Chih Chang	Dartmouth College
Jean-Lou De Carufel	University of Ottawa
Stephane Durocher	University of Manitoba
David Eppstein	University of California, Irvine
Will Evans	University of British Columbia
Meng He	Dalhousie University
Sheridan Houghten	Brock University
Shahin Kamali	York University
Anil Maheshwari	Carleton University
Debajyoti Mondal	University of Saskatchewan
Pat Morin	Carleton University
Rahnuma Islam Nishat (chair)	Brock University
Joseph O'Rourke	Smith College
Denis Pankratov	Concordia University
Michiel Smid	Carleton University
Venkatesh Srinivasan	Santa Clara University
Sue Whitesides	University of Victoria

Organizing Committee

Sheridan Houghten	Brock University
Rahnuma Islam Nishat (chair)	Brock University
Ke Qiu	Brock University
Justin Steepe	Brock University
Bastien Rivier	Brock University

Volunteers

Maysara Al Jumaily	Brock University
Kwok Tsz Yi	Brock University
Parth Chauhan	Brock University

Invited Speakers

Vašek Chvátal	Charles University (visiting) & Concordia University (emeritus)
Anna Lubiw	University of Waterloo
Mark Keil	University of Saskatchewan

CONFERENCE PROGRAM

Day 1: Wednesday July 17, 2024

Paul Erdős Memorial talk: The discrete mathematical charms of Paul Erdős *by Vašek Chvátal*

Session 1A

Session 1B

Session 2A

Session 2B

Day 2: Thursday July 18, 2024

Day 3: Friday July 19, 2024

Session 7B

361 List of Authors

Maintaining Light Spanners via Minimal Updates

David Eppstein * Hadi Khodabandeh †

Abstract

We study the problem of maintaining a lightweight bounded-degree $(1+\varepsilon)$-spanner of a dynamic point set in a d-dimensional Euclidean space, where $\varepsilon > 0$ and d are arbitrary constants. In our fully-dynamic setting, points are allowed to be inserted as well as deleted, and our objective is to maintain a $(1+\varepsilon)$-spanner that has constant bounds on its maximum degree and its lightness (the ratio of its weight to that of the minimum spanning tree), while minimizing the recourse, which is the number of edges added or removed by each point insertion or deletion. We present a fully-dynamic algorithm that handles point insertion with amortized constant recourse and point deletion with amortized $\mathcal{O}(\log \Delta)$ recourse, where Δ is the aspect ratio of the point set.

1 Introduction

Spanners are sparse subgraphs of a denser graph that approximate its shortest path distances. Extensive study has been made of *geometric spanners*, for which the dense graph is a complete weighted graph on a point set in d-dimensional Euclidean space, and where the weight of an edge (u, v) is simply the Euclidean distance between u and v. The approximation quality of a spanner is measured by its stretch factor t, where a t-spanner S is defined by the property that for every two vertices u and v in the graph, $d_S(u, v) \leq t \cdot d(u, v)$. Here d and d_S are respectively the Euclidean metric of dimension d and the shortest path metric induced by the spanner. In other words, the Euclidean distances are *stretched* by a factor of at most t in the spanner.

In this paper, we study the problem of maintaining $1+\varepsilon$-spanners under a dynamic model in which points are inserted and removed by an adversary and our goal is to minimize the recourse, which is the number of changes we make to the edge set of the spanner. The recourse should be distinguished from the time it takes us to calculate the changes we make, which might be larger; our use of recourse instead of update time is motivated by real-world networks, where making a physical change to the network is often more costly than the actual runtime of the algorithm that decides what changes need to be made.

As our main contribution in this paper, we construct a fully-dynamic spanner that maintains, at all times, a lightness and a maximum degree that are bounded by constants. Our maintenance regime achieves amortized constant recourse per point insertion, and amortized $\mathcal{O}(\log \Delta)$ recourse per point deletion. We state and prove our bounds in the following theorem:

Theorem 18 Our fully-dynamic spanner construction in d-dimensional Euclidean spaces has a stretch-factor of $1+\varepsilon$ and a lightness that is bounded by a constant. Furthermore, this construction performs an amortized $\mathcal{O}(1)$ edge updates following a point insertion, and an amortized $\mathcal{O}(\log \Delta)$ edge updates following a point deletion.

1.1 Related work

Geometric t-spanners have numerous applications in network design problems [15]. Finding a sparse lightweight t-spanner is the core of many of these applications. The existence of such spanners and efficient algorithms for constructing them have been considered under different settings and constraints [3, 12, 17]. In offline settings, where the point set is given as a whole to the algorithm, the prominent *greedy spanner* algorithm is well known for its all-in-one quality due to its optimal performance under multiple measures including sparsity (its number of edges), lightness (the weight of the spanner divided by the weight of the minimum spanning tree), and maximum degree [1, 4]. The output of the greedy spanner also has low crossing number in the plane and small separators and separator hierarchies in doubling metric spaces [8, 14]. However, in some applications, the points of an input set may repeatedly change as a spanner for them is used, and a static network would not accurately represent their distances. The dynamic model, detailed below, deal with these types of problems.

In the dynamic model, points are inserted or removed one at a time, and the algorithm has to maintain a t-spanner at all times. In this setting the algorithm is allowed to remove previous edges. For n points in d-dimensional Euclidean space, Arya, Mount, and Smid [2] designed a spanner construction with a linear number of edges and $\mathcal{O}(\log n)$ diameter under the assumption that a point to be deleted is chosen randomly from the point

*Department of Computer Science, University of California, Irvine, eppstein@uci.edu. Work funded by NSF grant CCF-2212129.

†Department of Computer Science, University of California, Irvine, khodabah@uci.edu. Work funded by NSF grant CCF-2212129.

set, and a point to be inserted is chosen randomly from the new point set. Bose, Gudmundsson, and Morin [5] presented a semi-dynamic $(1+\varepsilon)$-spanner construction with $\mathcal{O}(\log n)$ maximum degree and diameter. Gao, Guibas, and Nguyen [9] designed the deformable spanner, a fully-dynamic construction with $\mathcal{O}(\log \Delta)$ maximum degree and $\mathcal{O}(\log \Delta)$ lightness, where Δ is the aspect ratio of the point set, defined as the ratio of the length of the largest edge divided by the length of the shortest edge.

In the spaces of bounded doubling dimension, Roditty [16] provided the first dynamic spanner construction whose update time (and therefore recourse) depended solely on the number of points ($\mathcal{O}(\log n)$ for point insertion and $\tilde{\mathcal{O}}(n^{1/3})$ for point removal). This was later improved by Gottlieb and Roditty [11], who extended this result in doubling metrics and provided a better update time as well as the bounded-degree property. The same authors further improved this construction to have an asymptotically optimal insertion time (and therefore recourse) of $\mathcal{O}(\log n)$ under the algebraic decision tree model [10] but logarithmic lightness.

It is worth to mention that none of the work mentioned above in the dynamic setting achieve a sub-logarithmic lightness bound on their output. The problem of maintaining a light spanner in this setting has remained open until now.

2 Preliminaries and overview

In this section, we cover the notations as well as important definitions and facts that we use throughout the paper. We also provide an overview of what to expect in the upcoming sections and the methods we use to reach our bounds on the recourse.

Notation. We denote the current point set by V and its aspect ratio (as defined earlier) by Δ. We use the notations $\|e\|$ and $\|P\|$ for the Euclidean length of an edge e and a path P, respectively. We also refer to the Euclidean distance of two points u and v by $\|uv\|$ or $d(u,v)$, interchangeably. The notation $|E|$ is used when we are referring to the size of a set E. Also, for a spanner S, the weight of S is shown by $w(S)$.

2.1 Overview

We build our spanners on top of a hierarchical clustering $(\mathcal{T}, R)$ of the point set that we maintain dynamically as the point set changes over time. The tree $\mathcal{T}$ represents the parent-child relationship between the clusters, and the constant R specifies how cluster radii magnify on higher levels. Each cluster $\mathcal{C} \in \mathcal{T}$ is specified by a pair $\mathcal{C} = (p, l)$ where $p \in \mathbb{R}^d$ is one of the given points at the center of the cluster and $l \in \mathbb{Z}$ is the level of the cluster. The level of a cluster determines its radius, R^l. It is possible for the same point to be the center of multiple clusters, at different levels of the hierarchy.

We maintain our hierarchy so that after a point insertion, a cluster is added centered at the new point, and after a point deletion, each cluster with the deleted point as its center is removed. Meanwhile, we maintain a *separation* property on the hierarchy to help us build a sparse spanner. Additional edges of our sparse spanner connect pairs of clusters of the same level. Each such edge ensures that pairs of descendants of its endpoints have the desired stretch-factor. These edges form a bounded-degree graph on the clusters at each level, but this property alone would not ensure bounded degree for our whole spanner, because of points that center multiple clusters. Instead, we redistribute the edges of large degree points to derive a bounded-degree spanner.

Maintaining bounded lightness on the other hand is done through an iterative pruning process. We start by removing certain edges to decrease the weight of the spanner, which in turn might cause some other pairs that previously used the removed edge in their shortest paths to not meet the stretch bound of $1+\varepsilon$. We fix those pairs by adding an edge between them, which again increases the weight of the spanner. This causes a chain of updates that alternatively improve the stretch and worsen the weight of the spanner, or improve the weight and worsen the stretch of the spanner. We show that this sequence of updates, which we call *maintenance* updates, if performed properly and for the right pairs, will indeed not end in a loop, and even more strongly, will terminate after an amortized constant number of iterations. This will be covered in section 4.

The rest of this section includes the techniques we use for our light-weight spanner construction. We start with one of these techniques which is called the *bucketing* technique. Instead of enforcing the stretch bound and the lightness bound on the whole spanner, we partition its edges into a constant number of subsets and we enforce our criteria on these subsets. This partitioning is necessary for the purpose of our analysis.

Bucketing. We maintain a partition of the spanner edges into a constant number of subsets. As we mentioned before, our invariants are enforced on these subsets instead of the whole spanner. Let $C \gg c > 1$ be constants that we specify later. We partition the edges of the spanner into $k = \lceil \log_c C \rceil$ subsets, $S_0, S_1, \cdots, S_{k-1}$, so that for each set S_i and any pair of edges $e, f \in S_i$ such that $\|e\| \geq \|f\|$, one of the following two cases happen: (i) either $\|e\|/\|f\| < c$ or (ii) $\|e\|/\|f\| \geq C$. In other words, the edge lengths in the same set are either very close, or very far from each other.

Such partitioning can be maintained easily by assigning an edge e to the set with index $\text{index}(e) = \lfloor \log_c \|e\| \rfloor \bmod k$. We refer to this as the *index* of the edge e. We also define the *size* of an edge e as

size$(e) = \lfloor(\log_c\|e\|)/k\rfloor$. By definition, if index$(e) = i$ and size$(e) = j$, then $c^{kj+i} \leq \|e\| < c^{kj+i+1}$. We similarly define the index and the size for any pair (u, v) of vertices that are not necessarily connected in the spanner: index$(u, v) = \lfloor\log_c\|uv\|\rfloor \bmod k$, and size$(u, v) = \lfloor(\log_c\|uv\|)/k\rfloor$.

Invariants. In order to construct a light-weight spanner, we start from our sparse dynamic spanner construction. To distinguish the edges of our light spanner with the edges of our sparse spanner, we call the edges of our sparse spanner the *potential pairs*, since a carefully filtered set of those edges will make up our light-weight spanner. After bucketing the potential pairs, since we maintain the edges of each bucket separately, we must find per-bucket criteria that guarantee the the main properties we expect from our spanner: the stretch-factor and the lightness. We call these criteria the *invariants*. To make sure the union of the buckets meets the stretch bound, we generalize the notion of stretch factor to work on individual buckets and we call it Invariant 1.

- **Invariant 1.** For each pair of vertices $(u, v) \notin S_i$ with index i, there must exist a set of edges $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \ldots, e_l = (x_l, y_l)$ in S_i such that

$$\sum_{i=1}^{l}\|e_i\|+(1+\varepsilon)\left(\|ux_1\| + \sum_{i=1}^{l-1}\|y_ix_{i+1}\| + \|y_lv\|\right) < (1+\varepsilon)\|uv\|.$$

  In other words, u must reach v by a path of cost at most $(1+\varepsilon)\|uv\|$ where the cost of every edge $e \in S_i$ is $\|e\|$ and the cost of every edge $e \notin S_i$ is $(1+\varepsilon)\|e\|$.

Lemma 1 *If Invariant 1 holds for all S_i, then $S = \bigcup_{i=0}^{k-1} S_i$ is a $(1+\varepsilon)$-spanner.*

Proof. [Proof of Lemma 1] Let (u, v) be a pair of vertices. We find a $(1+\varepsilon)$-path between u and v using edges in S. Let $i = \text{index}(u, v)$. By Invariant 1 there exists a set of edges $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \ldots, e_l = (x_l, y_l)$ in S_i such that

$$\sum_{i=1}^{l}\|e_i\|+(1+\varepsilon)\left(\|ux_1\| + \sum_{i=1}^{l-1}\|y_ix_{i+1}\| + \|y_lv\|\right) < (1+\varepsilon)\|uv\|.$$

Consider the path $P = ux_1y_1x_2y_2\cdots x_ly_lv$ between u and v. We call this path the *replacement path* for (u, v). The edges $x_1y_1, x_2y_2, \ldots, x_ly_l$ are present in S_i (and therefore present in S) but the other edges of the replacement path are missing from S_i. A similar procedure can be performed on the missing pairs recursively to find and replace them with their corresponding replacement paths. This recursive procedure yields a $(1+\varepsilon)$-path for (u, v) and it terminates because the length of each missing edge in a replacement path is smaller than the length of the edge that is being replaced (otherwise Invariant 1 would not hold). □

Furthermore, we bound the weight of the spanner by ensuring the second invariant, which is the leapfrog property on S_i. [7]

- **Invariant 2.** Let $(u, v) \in S_i$. For every subset of edges $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \ldots, e_l = (x_l, y_l)$ in S_i the inequality

$$\sum_{i=1}^{l}\|e_i\|+(1+\varepsilon)\left(\|ux_1\| + \sum_{i=1}^{l-1}\|y_ix_{i+1}\| + \|y_lv\|\right) > (1+\varepsilon')\|uv\|$$

  holds, where $\varepsilon' < \varepsilon$ is a positive constant. In other words, u should not be able to reach v by a (short) path of cost $(1+\varepsilon')\|uv\|$, where the edge costs are the same as in Invariant 1.

The leapfrog property leads to a constant upper bound on the lightness of S_i, for each $0 \leq i < k$. And since the weight of the minimum spanning tree on the end-points of each S_i is at most a constant factor of the weight of the minimum spanning tree on the whole point set, this implies a constant upper bound on the lightness of the spanner $S = \bigcup_{i=0}^{k-1} S_i$. As well as the weight bound, we prove, in the following lemma, that Invariant 2 implies a similar result to the packing lemma, but for the number of edges on the same level.

Lemma 2 (Edge packing) *Let E be a set of edges (segments) with the same index and the same level that is consistent with Invariant 2. Also, assume that E is contained in a ball of radius R, and the minimum edge size in E is r. Then*

$$|E| < C_1(R/r)^{2d}$$

where $C_1 = (2(1+\varepsilon)/\varepsilon')^{2d}d^d$ is a constant.

Proof. [Proof of Lemma 2] A simple observation is that for any two segments (u, v) and (y, z) in E we must have

$$\max(\|uy\|, \|vz\|) > \frac{\varepsilon'}{2(1+\varepsilon)} \cdot r$$

because otherwise, assuming that $\|uv\| \geq \|yz\|$, for the pair (u, v) and the sequence $e_1 = (y, z)$, the left hand side of the inequality in Invariant 2 would be at most

$$2(1+\varepsilon) \cdot \frac{\varepsilon'}{2(1+\varepsilon)} \cdot r + \|yz\| \leq (1+\varepsilon')\|uv\|$$

contradicting the fact that E is consistent with Invariant 2. Thus, given a covering of a ball of radius R with

M balls of radius $r' = \frac{\varepsilon'}{2(1+\varepsilon)} \cdot r$, every segment in E has its endpoints in a unique pair of balls, otherwise Invariant 2 will be compromised. Hence, $|E| \leq M^2$. A simple calculation yields a covering with $M < (2(1+\varepsilon)/\varepsilon')^d d^{d/2}(R/r)^d$ balls. □

We can simplify the two invariants by defining a distance function d_i^* over the pairs of vertices,

Definition 1 *Let S_i^* be a complete weighted graph over the vertices such that the weight of an edge e in S_i^* is defined as*

$$w(e) = \begin{cases} \|e\| & \text{if } e \in S_i \\ (1+\varepsilon)\|e\| & \text{if } e \notin S_i \end{cases}$$

We define an extended path *between u and v in S_i as a path between u and v in S_i^* that only uses edges (y,z) where $\text{size}(y,z) < \text{size}(u,v)$. We also define the length of an extended path as the sum of its edge weights in S_i^*. Finally, we define $d_i^*(u,v)$ as the length of the shortest extended path between u and v.*

Using this new distance function we can rephrase the two invariants as follows.

- **Invariant 1.** For every pair $(u,v) \notin S_i$ with $\text{index}(u,v) = i$, we have $d_i^*(u,v) < (1+\varepsilon)d(u,v)$.
- **Invariant 2.** For every pair $(u,v) \in S_i$, we have $d_i^*(u,v) > (1+\varepsilon')d(u,v)$.

It is worth noting that these forms are not exactly equivalent to the previous forms, as we are only considering paths of lower level edges in the definition of d_i^*, while a short path in the spanner could potentially contain an edge of the same level. This provides a stronger variation of Invariant 1, which still implies a $1+\varepsilon$ stretch for the spanner. However, this change weakens Invariant 2. But as we will see, a careful addition of the same-level edges can prevent any possible violations of Invariant 2 that could be caused by this new form.

Maintaining the invariants. The quality of our light-weight dynamic spanner depends on the two invariants we introduced above, and an update like a point insertion or removal could cause one of them to break, if not both. Therefore, we establish a procedure that addresses the inconsistencies and enforces the invariants to hold at all times.

The procedure for fixing a violation of Invariant 1 is straightforward: as long as there exists a pair (u,v) that violates Invariant 1 for its corresponding subset S_i, add an appropriate potential pair to S_i that connects an ancestor of u to an ancestor of v in the hierarchy $\mathcal{T}$. This resolves the inconsistency for (u,v) if the ancestors are chosen properly, but it might cause other pairs to violate Invariant 2 because of this edge addition. We will prove that if certain criteria are met, there would be no side effect on the same-level pairs and the addition can only result in a constant amortized number of inflicted updates on higher level pairs.

Fixing a violation of Invariant 2, on the other hand, is more tricky. After we remove the violating edge (u,v) from its subset S_i, the effect on higher level pairs would be similar to the previous case, but removing (u,v) might cause multiple updates on the same level, which in turn cascade to higher levels. We therefore analyze the removal of (u,v) together with the subsequent additions of same-level edges that aim to fix the incurred violations of Invariant 1, and we prove that a constant amortized bound on the number of inflicted updates on higher level pairs would still hold. We get to the details of our maintenance updates in section 4.3.

Amortized analysis. We analyze the effects of an update (edge addition and removal) on higher level pairs using a potential function, for each S_i separately. We define our potential function over the potential pairs in S_i. The change in the potential function shows how much a pair is close to violating one of the invariants. The higher the potential, the closer the pair is to violating the invariants. This enables us to assign a certain amount of credit to each update, that can be used to pay for the potential change of the updated pair and the affected pairs, which in turn results on an amortized upper bound on the number of edge updates in the future. Therefore, for a potential pair (u,v) with index i and following an update in S_i,

- if $(u,v) \in S_i$ and $d_i^*(u,v)$ decreases, or
- if $(u,v) \notin S_i$, and $d_i^*(u,v)$ increases,

we increase the potential of the pair (u,v) to account for its future violation of the invariants.

More specifically, we define the potential function $p_i(u,v)$ of a potential pair (u,v) in S_i as

$$p_i(u,v) = \begin{cases} (1+\varepsilon) - \frac{d_i^*(u,v)}{d(u,v)} & \text{if } (u,v) \in S_i \\ C_\phi \cdot \left(\frac{d_i^*(u,v)}{d(u,v)} - (1+\varepsilon')\right) & \text{if } (u,v) \notin S_i \text{ and } \text{index}(u,v) = i \end{cases}$$

where $C_\phi > 1$ is a positive constant coefficient that we specify later. This implies that if $p_i(u,v) < \varepsilon - \varepsilon'$, then both invariants would hold for the pair (u,v) (in S_i). Based on this observation, we define a potential function on S_i in the following way,

$$\Phi_i = \sum_{(u,v) \in \mathcal{P}_i \cup S_i} p_i(u,v)$$

where $\mathcal{P}_i$ is the set of potential pairs with index i. We simply define the potential of the whole spanner as

$$\Phi = \sum_i \Phi_i$$

We add another term to this potential function later in section 4 to account for future edges between the existing nodes.

$$\Phi^* = \Phi + \frac{p_{max}}{2} \cdot \sum_{i=1}^{n} (D_{max} - \deg_{\mathcal{S}_1}(v_i))$$

We first prove some bounds on Φ but we ultimately use the adjusted potential function Φ^* to prove our amortized bounds on the number of updates. In the remainder of this paper, we specify our sparse and lightweight construction in more details, and we will provide our bounds on the recourse in each case separately.

3 Sparse spanner

In this section, we introduce our dynamic construction for a sparse spanner with constant amortized recourse per point insertion and $\mathcal{O}(\log \Delta)$ recourse per point deletion. We build our spanner on top of a hierarchical clustering that we design early in this section.

Krauthgamer and Lee [13] showed how to maintain such hierarchical structures in $\mathcal{O}(\log \Delta)$ update time by maintaining ε-nets. However, this hierarchy is not directly applicable to our case since a point can appear $\log \Delta$ times on its path to root, which would imply a $\mathcal{O}(\log \Delta)$ bound on the degree of the spanner instead of a constant bound. Cloe and Gottlieb [6] improved the update time of this hierarchy to $\mathcal{O}(\log n)$. Gottlieb and Roditty [10] later introduced a new hierarchical construction with the same update time for their fully-dynamic spanner, which also satisfied an extra close-containment property. Here, we introduce a simpler hierarchy that suits our needs and does not require the close-containment property. Our hierarchy performs constant cluster updates for a point insertion and $\mathcal{O}(\log \Delta)$ cluster updates for a point deletion.

Our hierarchy consists of a pair $(\mathcal{T}, R)$ where $\mathcal{T}$ is a rooted tree of clusters and $R > 0$ is a constant. Every cluster $\mathcal{C} \in \mathcal{T}$ is associated with a center $c(\mathcal{C}) \in V$ and a level $l(\mathcal{C}) \in \mathbb{Z}$. The level of a cluster specifies its radius; $\mathcal{C}$ *covers* a ball of radius $R^{l(\mathcal{C})}$ around $c(\mathcal{C})$. We denote the *parent* of $\mathcal{C}$ in $\mathcal{T}$ by $p(\mathcal{C})$. The root of $\mathcal{T}$, denoted by $\mathcal{T}.root$, is the only cluster without a parent. Furthermore, the level of a parent is one more than of the child, i.e. $l(p(\mathcal{C})) = 1 + l(\mathcal{C})$, for all $\mathcal{C} \in \mathcal{T}$ except the root. A parent must cover the centers of its children.

Besides these basic characteristics, we require our hierarchy to satisfy the separation property at all times. This property states that the clusters at the same level are separated by a distance proportional to their radii,

Definition 2 (Separation property) *For any pair of same-level clusters $\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{T}$ on level j,*

$$d(c(\mathcal{C}_1), c(\mathcal{C}_2)) > R^j$$

Each point at the time of insertion creates a single cluster centered at the inserted point, and during the future insertions, might have multiple clusters with different radii centered at it. In fact, each point could have clusters centered at it in at most $\mathcal{O}(\log \Delta)$ levels. At the time of deletion, any cluster that is centered at the deleted point will be removed.

Our clusters are of two types: explicit clusters and implicit clusters. Explicit clusters are the ones we create manually during our maintenance steps. Implicit clusters are the lower level copies of the explicit clusters that exist in the hierarchy even though we do not create them manually. Therefore, if a cluster $\mathcal{C} = (p, l)$ is created in the hierarchy at some point, we implicitly assume clusters (p, i) for $i < l$ exist in the hierarchy after this insertion, and they are included in their corresponding $\mathcal{T}_i$ as well. We maintain the separation property between all clusters, including the implicit ones. We use these implicit clusters for constructing our spanner.

3.1 Maintaining the hierarchy

We initially start from an empty tree $\mathcal{T}$ and a constant R that we specify later.

Point insertion. Let $\mathcal{T}_i$ be the set of clusters with level i, i.e. $\mathcal{T}_{\text{size}(\mathcal{T}.root)}$ only contains the root, $\mathcal{T}_{\text{size}(\mathcal{T}.root)-1}$ contains root's children, etc. Upon the insertion of a point p, we look for the lowest level (between explicit clusters) i that p is covered in $\mathcal{T}_i$. We insert $\mathcal{C} = (p, i-1)$ into the hierarchy. Since p is covered in T_i, we can find a cluster $\mathcal{C}' = (p, i)$ that covers p and assign it as the parent of $\mathcal{C}$ (Algorithm 1).

In the case that p is not covered in any of the levels in $\mathcal{T}$, which we handle by replicating the root cluster from above until it covers the new point, then the insertion happens the same way as before.

Algorithm 1 Inserting a point to the hierarchy.

```
1: procedure INSERT-TO-HIERARCHY(T, R, p)
2:     if |T| = 0 then
3:         Add a root cluster C = (p, 0) to T.
4:         return C
5:     Let i be the lowest level in T.
6:     while T_i does not cover p do
7:         Increase i by 1.
8:         if i > size T.root then
9:             Create a new cluster C = (T.root, size(T.root) + 1).
10:            Make C the new root of the hierarchy.
11:            The old root becomes a child of C.
12:    Let C' be a cluster in T_i that covers p.
13:    Create a cluster C = (p, size(C') − 1) and add it as a child
   of C'.
```

The basic characteristics of the hierarchy hold after an insertion. We now show that the separation property holds after the insertion of a new cluster $\mathcal{C} = (p, l)$. Assume, on the contrary, that there exists a cluster $\mathcal{C}' = (q, l)$ that $(\mathcal{C}, \mathcal{C}')$ violates the separation property.

$\mathcal{C}$ is inserted on level l, thus p is not covered by $\mathcal{T}_l$. According to the assumption, $d(q,p) \leq R^l$, meaning that $\mathcal{C}'$ covers p. This contradicts the fact that $\mathcal{T}_l$ does not cover p since $\mathcal{C}' \in \mathcal{T}_l$. A similar argument shows that the separation property holds for the implicit copies of $\mathcal{C}$ as well.

Point deletion. Upon the deletion of a point p, we remove all the clusters centered at p in the hierarchy. The clusters centered at p create a chain in $\mathcal{T}$ that starts from the lowest level explicit copy of p and ends at the highest level copy. We remove this chain level by level, starting from the lowest level cluster $\mathcal{C} = (p,l)$ that is centered at p. Upon the removal of $\mathcal{C}$, we loop over children of $\mathcal{C}$ one by one, and we try to assign them to a new parent. If we find a cluster on level $l+1$ that covers them, then we assign them to that cluster, otherwise we replicate them on one level higher and we continue the process with the remaining children. After we are done with (p,l), we repeat the same process with $(p,l+1)$, until no copies of p exist in the hierarchy (Algorithm 2).

Algorithm 2 Deleting a point from the hierarchy.

```
1: procedure DELETE-FROM-HIERARCHY(T, R, p)
2:     Let C = (p, l) be the lowest level (explicit) cluster centered at p.
3:     Delete C from T and mark its children.
4:     while there exists a marked cluster on level l − 1 do
5:         Let C' = (q, l − 1) be a marked cluster.
6:         Find a cluster C'' on level l that covers q.
7:         if such cluster exists then
8:             Assign C'' as the parent of C' and unmark C'.
9:         else
10:            Create C'' = (q, l) and make it the parent of C'.
11:            Mark C'' and unmark C'.
12:    if there still exists a marked cluster in T then
13:        Increase l by one and repeat the while loop above.
```

Again, the basic characteristics of the hierarchy hold after a deletion. We need to show that the separation property still holds. Immediately after removing the cluster (p,l) the separation property obviously holds. After re-assigning a marked child to another parent the property still holds since no cluster has changed in terms of their center or level. If a marked child is replicated on level $l+1$, it means that there was no cluster covering it on this level, otherwise it would have been assigned as its new parent. Therefore, the separation property holds after the replication on level $l+1$. We will prove more properties of our hierarchy later on when we define the spanner.

3.2 The initial spanner

Our initial spanner is a sparse spanner that is defined on the hierarchy $\mathcal{T}$ and it has bounded cluster degree but not bounded point degree. The reason that a bounded degree on the clusters would not imply a bounded degree on the point set is that every point could have multiple clusters centered at it, each of which have a constant number of edges connected to them. This would cause the degree of the point to get as large as $\Omega(\log \Delta)$. Later we will fix this issue by assigning edges connected to large degree points to other vertices.

The initial spanner consists of two types of edges. The first type that we already mentioned before, is the edges that go between clusters of the same level. These edges guarantee a short path between the descendants of the two clusters, similar to a spanner built on a well-separated pair decomposition. And the second type is the parent-child edges, that connect every node to its children. The edge weight between two clusters is the same as the distance between their centers.

We define the spanner formally as follows,

Definition 3 (Initial spanner) *Let $(\mathcal{T}, R)$ be a hierarchy that satisfies the separation property. We define our sparse spanner $\mathcal{S}_0$ to be the graph on the nodes of $\mathcal{T}$ that contains the following edges,*

- *Type I. Any pair of centers p and q whose clusters are located on the same level and $d(p,q) \leq \lambda \cdot R^l$ are connected together. Here, λ is a fixed constant.*
- *Type II. Any cluster center in $\mathcal{T}$ is connected to the centers of its children in $\mathcal{T}$.*

Note that the implicit clusters are also included in this definition. Meaning that if two implicit same-level clusters are close to each other then there would be an edge of type I between them. We show that the spanner $\mathcal{S}_0$ has a bounded stretch.

Lemma 3 (Stretch-factor) *For large enough $\lambda = \mathcal{O}(\varepsilon^{-1})$ the stretch-factor of $\mathcal{S}_0$ would be bounded from above by $1+\varepsilon$.*

Proof. Let p and q be two points in the point set, and also let $\mathcal{C} = (p,l)$ and $\mathcal{C}' = (q,l')$ be the highest level clusters in $\mathcal{T}$ that are centered at p and q, respectively. By symmetry, assume $l \geq l'$. If $d(p,q) \leq \lambda \cdot R^{l'}$, then there is an edge between the (possibly implicit) cluster (p,l') and $\mathcal{C}'$. This edge connects p and q together, therefore the stretch would be equal to 1 for this pair. If $d(p,q) > \lambda \cdot R^{l'}$, we perform an iterative search for such shortcut edge. Start with $\mathcal{C} = (p,l')$ and $\mathcal{C}' = (q,l')$ and every time that the inequality $d(p,q) \leq \lambda \cdot R^{l'}$ is not satisfied set $\mathcal{C}$ and $\mathcal{C}'$ to their parents and set $l' = l'+1$ and check for the inequality again. We show that the inequality eventually will be satisfied. Let p_i and q_i be the centers of $\mathcal{C}$ and $\mathcal{C}'$ on the i-th iteration of this iterative process ($i = 1, 2, \dots$), and let l' have its initial value before any increments. We have $d(p_{i+1}, p_i) \leq R^{l'+i}$ and $d(q_{i+1}, q_i) \leq R^{l'+i}$. By the triangle inequality,

$$d(p_{i+1}, q_{i+1}) \leq d(p_{i+1}, p_i) + d(p_i, q_i) + d(q_{i+1}, q_i) \leq 2 \cdot R^{l'+i} + d(p_i, q_i)$$

Denote the ratio $d(p_i, q_i)/R^{l'+i-1}$ by x_i. We have,

$$x_{i+1} \leq 2 + \frac{x_i}{R}$$

Therefore, x_i is roughly being divided by R on every iteration and it stops when $x_i \leq \lambda$. We can easily see that the loop terminates and the value of x_i after the termination would be greater than λ/R. This particularly shows that the edge between $\mathcal{C}$ and $\mathcal{C}'$ is a long shortcut edge when λ is chosen large enough, since its length is more than λ/R times the radius of the centers it is connecting.

Now we show that this shortcut edge would be good enough to provide the $1+\varepsilon$ stretch factor for the initial points, p and q. Note that because of the parent-child edges, p can find a path to q by traversing p_is in the proper order and using edge between p_i and q_i and traversing back to q. We show that the portion of the path from p to p_i (and similarly from q to q_i) is at most $\frac{R^{l'+i}-1}{R-1}$. We prove it only for p, the argument for q is similar. Note that if the termination level $l'+i \leq l$ then $p_i = p$ and this path length from p to p_i would be 0, confirming our claim for p. Therefore, we assume the termination level is above the level of p. The length of the path from p to p_i that only uses type II edges would be at most

$$R^{l+1} + \cdots + R^{l'+i} < \frac{R^{l'+i+1}-1}{R-1}$$

Thus the length of the path from p to q would be at most

$$2 \cdot \frac{R^{l'+i+1}-1}{R-1} + d(p_i, q_i)$$

On the other hand, by the triangle inequality,

$$d(p,q) \geq d(p_i, q_i) - 2 \cdot \frac{R^{l'+i+1}-1}{R-1}$$

Finally, the stretch-factor of this path would be at most

$$\frac{2 \cdot \frac{R^{l'+i+1}-1}{R-1} + d(p_i, q_i)}{d(p_i, q_i) - 2 \cdot \frac{R^{l'+i+1}-1}{R-1}}$$

A simple calculation yields that this fraction is less than $1+\varepsilon$ when $\lambda = 2(2+\varepsilon)\varepsilon^{-1}R = \mathcal{O}(\varepsilon^{-1})$. □

Next, we show that the degree of every cluster in $\mathcal{S}_0$ is bounded by a constant. Note that this does not imply a bounded degree on every point, since a point could be the center of many clusters.

Lemma 4 (Degree bound) *The degree of every cluster in $\mathcal{S}_0$ is bounded by $\mathcal{O}(\varepsilon^{-d})$.*

Proof. We first prove that the type I degree of every cluster $\mathcal{C} = (p, l)$ is bounded by a constant. Let $\mathcal{C}' = (q, l)$ be a cluster that has a type I edge to $\mathcal{C}$. This means that $d(p,q) \leq \lambda \cdot R^l$. By the separation property, $d(p,q) > R^l$. Thus, by the packing lemma there are at most

$$d^{d/2}\lambda^d = \mathcal{O}(\varepsilon^{-d})$$

type I edges connected to $\mathcal{C}$. The last bound comes from the fact that a choice of $\lambda = \mathcal{O}(\varepsilon^{-1})$ would be enough to have a bounded stretch.

Now we only need to show that the parent-child edges also add at most a constant degree to every cluster, which is again achieved by the packing lemma. Because the children of this cluster are located in a ball of radius R^l around its center, p, and they are also pair-wise separated by a distance of at least R^{l-1}, we can conclude that the number of children of $\mathcal{C}$ would be upper bounded by $d^{d/2}R^d = \mathcal{O}(1)$. □

Representative assignment. So far we showed how to build a spanner that has a bounded degree on each cluster and the desired stretch-factor of $1+\varepsilon$. But this spanner does not have a degree bound on the actual point set and that is a property we are looking for. Here, we show how to reduce the load on high degree points and distribute the edges more evenly so that the bounded degree property holds for the point set as well.

The basic idea is that for every cluster $\mathcal{C}$ in the hierarchy, we pick one of lower level clusters, say $\mathcal{C}'$, to be its *representative* and play its role in the final spanner, meaning that all the spanner edges connecting $\mathcal{C}$ to other clusters will now connect $\mathcal{C}'$ to those clusters after the re-assignment. This re-assignment will be done for every cluster in the hierarchy until every cluster has a representative. Only then we can be certain that the spanner has a bounded degree on the current point set. Since by Lemma 4 the degree of every center is bounded by a constant, we only need to make sure that every point is representing at most a constant number of clusters in the hierarchy.

First, we define the level of a point p, denoted by $\text{size}(p)$ to be the level of the highest level cluster that has p as its center, i.e. $\text{size}(p) = \max_{(p,l)\in\mathcal{T}} l$.

Definition 4 (Representative assignment) *Let $\mathcal{T}$ be a hierarchy. We define the representative assignment of $\mathcal{T}$ to be a function $\mathcal{L}$ that maps every cluster $\mathcal{C} = (p, l)$ of $\mathcal{T}$ to a point q in the point set such that $l \geq \text{size}(q)$ and $d(p,q) \leq R^l$. We say $\mathcal{L}$ has bounded repetition b if $|\mathcal{L}^{-1}(q)| \leq b$ for every point q.*

Connecting the edges between the representatives instead of the actual centers would give us our bounded-degree spanner.

Definition 5 (Bounded-degree spanner) *Define the spanner $\mathcal{S}_1$ to be the spanner connecting the pair $(\mathcal{L}(\mathcal{C}), \mathcal{L}(\mathcal{C}'))$ for every edge $(\mathcal{C}, \mathcal{C}') \in \mathcal{S}_0$.*

Now we show that this re-assignment of the edges would not affect the stretch-factor and the degree bound significantly if the clusters are small enough, or equivalently, λ is chosen large enough.

Lemma 5 (Stretch-factor) *For large enough $\lambda = \mathcal{O}(\varepsilon^{-1})$ and any representative assignment $\mathcal{L}$ the stretch-factor of $\mathcal{S}_1$ would be bounded from above by $1+\varepsilon$.*

Proof. The proof works in a similar way to the proof of Lemma 3. A shortcut edge would still provide a good path between two clusters even after its end points are replaced by their representatives. The path from a p to p_i will be doubled at most since a representative could be as far as a child from the center of a cluster. Therefore, the stretch-factor of the path between p and q will be

$$\frac{4 \cdot \frac{R^{l'+i+1}-1}{R-1} + d(p_i, q_i)}{d(p_i, q_i) - 4 \cdot \frac{R^{l'+i+1}-1}{R-1}}$$

Again, this fraction is less than $1+\varepsilon$ when $\lambda = 4(2+\varepsilon)\varepsilon^{-1}R = \mathcal{O}(\varepsilon^{-1})$. □

To construct a bounded-repetition representative assignment we pay attention to the neighbors of lower level copies of a cluster. Let $\mathcal{C} = (p, l)$ be a cluster that we want to find a representative for. As we mentioned before, (p, l') exists in the hierarchy for all $l' < l$. If l' is small enough, i.e. $l' < l - \log_R \lambda$, then the neighbors of (p, l') will be located within a distance $\lambda \cdot R^{l'} = R^l$ of p, making them good candidates to be a representative of $\mathcal{C}$. Therefore, having more neighbors on lower levels means having more (potential) representatives on higher levels. This is how we assign the representatives.

We define a chain to be a sequence of clusters with the same center that form a path in $\mathcal{T}$. We divide a chain into blocks of length $\log_R \lambda$. The best way to do this so that maintaining it dynamically is easy is to index the clusters in a chain according to their levels and gather the same indices in the same block. We define the block index of a cluster in a chain to be $\lfloor l / \log_R \lambda \rfloor$, where l is the level of the cluster. The clusters in a chain that have the same index form a block.

The first observation is that if we are given two non-consecutive blocks in the same chain, we can use the neighbors of the lower level block as representatives of the higher level block. This is the key idea to our representative assignment, which we call *next block assignment*. In this assignment, we aim to represent higher level points with lower level points. Let p be a point and $P_1, P_2, \ldots, P_k$ be all the blocks of the chain that is centered at p in $\mathcal{T}$, ordered from top to bottom (higher level blocks to lower level blocks). We say a block is empty if the clusters in the block have no neighbors in $\mathcal{T}$. We say the block is non-empty otherwise. We make a linked list $\mathcal{L}_0$ of all the even indexed non-empty blocks, and a separate linked list $\mathcal{L}_1$ for all the odd indexed non-empty blocks. For every element of $\mathcal{L}_0$ we pick an arbitrary neighbor cluster of its block in $\mathcal{L}_0$ (because the blocks are non-empty such neighbors exists), and we assign that neighbor to be the representative of the clusters in that element. More specifically, let B_i be a block in $\mathcal{L}_0$, and let B_{i+1} be the next block in $\mathcal{L}_0$. Let $\mathcal{C}$ be an arbitrary cluster in B_{i+1} that has a neighbor. This cluster exists, since B_{i+1} is a non-empty block. Let q be the center of a neighbor of $\mathcal{C}$. We assign $\mathcal{L}(\mathcal{C}') = q$ for all $\mathcal{C}' \in B_i$. The same approach works for $\mathcal{L}_1$. This assigns a representative to every block in the chain, except the last block in $\mathcal{L}_0$ and $\mathcal{L}_1$. We assign p itself to be the representative of the clusters in these blocks.

Now we show that this assignment has bounded repetition. First, we show that our assignment only assigns lower level points to be representatives of higher level points.

Lemma 6 *Let p and q be two points in the point set and let $\text{size}(p) > \text{size}(q)$. In the next block assignment q would never be represented by p.*

Proof. Assume, on the contrary, that q is represented by p. Therefore, there exists two same-parity cluster blocks in the chain centered at q that a cluster centered at p is connected to the lower block. Let $\mathcal{C} = (p, l)$ and $\mathcal{C}' = (q, l')$ be the highest clusters centered at p and q, respectively. Since the connection between p and q is happening somewhere on the third block or lower on the chain centered at q, we can say that $d(p, q) < \lambda \cdot R^{l' - \log_R \lambda} = R^{l'}$. This means that the separation property does not hold for the lower level copy of $\mathcal{C}$, (p, l'), and $\mathcal{C}'$, which is a contradiction. □

Now that we proved that points can only represent higher level points in our assignment, we can show the bounded repetition property.

Lemma 7 (Bounded repetition) *The next block assignment $\mathcal{L}$ described above has bounded repetition.*

Proof. We show that every point represents at most a constant number of clusters. First, note that the two bottom clusters of the two block linked lists have a constant number of clusters in them (to be exact, $2 \log_R \lambda$ clusters maximum). So we just need to show that the number of other clusters that are from other chains and assigned to the point are bounded by a constant. Let p be an arbitrary point and let $\mathcal{C} = (p, l)$ be the highest level cluster centered at p. According to the previous lemma, any point q that has a cluster $\mathcal{C}' = (q, l')$ that $\mathcal{L}(\mathcal{C}') = p$ must have a higher level than p. Therefore, there exists a lower level copy of q on level l. Also, the distance between p and q is bounded by $\lambda \cdot R^l$ since p and q are connected on a level no higher than l (remember that we only represent our clusters with their lower level

neighbors). Now we can use the packing lemma, since all such points q have a cluster centered at them on level l and therefore separated by a distance of R^l. By the packing lemma, the number of such clusters would be bounded by $d^{d/2}\lambda^d$ such points. So the repetition is at most $b = d^{d/2}\lambda^d + 2$. □

Corollary 8 *The spanner $\mathcal{S}_1$ has bounded degree.*

3.3 Maintaining the spanner

So far we showed $\mathcal{S}_1$ has bounded stretch and bounded degree. Here we show that we can maintain $\mathcal{S}_1$ with $\mathcal{O}(1)$ amortized number of updates after a point insertion and $\mathcal{O}(\log \Delta)$ amortized number of updates after a point deletion. We know how to maintain the hierarchy from earlier in this section. Therefore, we just explain how to update the spanner, which includes maintaining our representative assignments dynamically.

Point insertion. We prove the amortized bound by assigning credits to each node, and using the credit in the future in the case of an expensive operation. Let D_{max} be the degree bound we proved for $\mathcal{S}_1$. When a new point is added to the spanner, we assign D_{max} credits to it.

We analyze the edge addition and removals that happen after the insertion of a point p in the spanner. Note that although only one explicit cluster is added to $\mathcal{T}$ after the insertion, there might be many new edges between the implicit (lower level) copies of the new cluster and other clusters that existed in $\mathcal{T}$ beforehand. We need to show that these new edges do not cause a lot of changes on the spanner after the representative assignment phase.

First, we analyze the effect of addition of p on points q that size$(p) >$ size(q). Similar to the proof of Lemma 6, we can show that any edges between the chain centered at p and the chain centered at q will be connected to the top two cluster blocks of the chain centered at q. This means that these edges will have no effect on the assignment of other clusters in the chain centered at q, because each non-empty block is represented by some neighbor of the next non-empty same-parity block, and the first two blocks, whether they are empty or not, will not have any effect on the rest of the assignment. Therefore, no changes will occur on the representatives of q and therefore the edges that connect these representatives together will remain unchanged.

The addition of p as we mentioned, would cause the addition of some edges in the spanner $\mathcal{S}_1$, that we pay for using the constant amount of credit stored on the endpoints of those edges. Therefore, we are not spending more than constant amount of amortized update for this case.

Second, we analyze the effect of addition of p on points q that size$(p) \leq$ size q. The outcome is different in this case. Similar to the previous case we can argue that any edge between the chain centered at p and the chain centered at q must be connected to the top two blocks of the chain centered at p, but they could be connected to anywhere relative to the highest cluster centered at q. This means that they could add a non-empty block in the middle of the chain centered at q. If this happens, then the assignment of the previous non-empty same-parity block changes and also the new non-empty block will have its own assignment. This translates into a constant number of changes (edge additions and removals) on the spanner $\mathcal{S}_1$ per such point q. We earlier in Lemma 7 proved that there is at most a constant number of such clusters. This shows that there would be at most a constant number of changes on the spanner $\mathcal{S}_1$ from higher level points.

Finally, we can conclude that overall the amortized recourse for insertion is bounded by a constant, since in the first case we could pay for the changes using the existing credits, and in the second case we could pay for the changes from our pocket.

Point deletion. After a point deletion, all the clusters centered at that point will be removed from the hierarchy, and a set of replication to higher levels would happen to some clusters to fix the hierarchy after the removal. It is easy to see that the number of cluster changes (including removal and replication) would be bounded by a constant. Each cluster change would also cause a constant number of changes on the edges of the spanner $\mathcal{S}_0$. Note that a cluster removal can introduce an empty block to at most a constant number of higher level points and a cluster replication can also introduce an empty block to at most a constant number of higher level points. Therefore, the changes on the representative assignments would be bounded by a constant after a single cluster update. Since we have at most $\mathcal{O}(\log \Delta)$ levels in the hierarchy, each of which having at most a constant number of cluster updates, overall we would have at most $\mathcal{O}(\log \Delta)$ number of edge changes on $\mathcal{S}_1$. After the removal, we assign full D_{max} credit to any node that is impacted by the removal. This would make sure we have enough credits for the future additions.

4 Light spanner

In section 3 we discuss how we maintain our hierarchical clustering and how we construct and maintain a sparse spanner on top of this hierarchy so that each point insertion makes at most $\mathcal{O}(1)$ changes on the spanner and each point deletion makes at most $\mathcal{O}(\log \Delta)$ changes on the spanner.

In this section, we introduce our techniques for maintaining a light spanner that has a constant lightness bound on top of all the properties we had so far. In our main result in this section we show that maintaining the lightness in our case is not particularly harder

than maintaining the sparsity, meaning that it would not require asymptotically more changes than a sparse spanner would.

We first analyze the effect of point insertion or deletion on the potential functions we defined earlier in section 2. Then we introduce our maintenance updates and we show our bounds on the recourse of a light spanner.

4.1 Bounding the potential function

In this section we analyze the behavior of our potential functions, after a point insertion and a point deletion. These bounds will later help us prove the amortized bounds on the recourse. We refer to section 2 for the definition of the potential function.

Single edge update. We start with a simple case of bounding the potential function after a single edge insertion, then we consider a single edge deletion, and finally we extend our results to point insertions and deletions. We assume the pair that we insert to or delete from the spanner is an arbitrary pair from the set of potential pairs, because we only deal with potential pairs in our light spanner.

First, we consider a single edge insertion. We divide the analysis into two parts: the effect of the insertion of the potential pair onto the same level potential pairs, and the effect of the insertion onto higher level potential pairs. Recall that the level of a pair was defined in section 2. We show that the edges of the same level satisfy a separation property, meaning that two edges in the same bucket cannot have both their endpoints close to each other.

Lemma 9 (Edge separation) *Let (u, w) and (y, z) be two potential pairs in the same bucket. Assuming that (u, w) and (y, z) are not representing clusters from the same pair of chains in $\mathcal{T}$,*

$$\max\{d(u, y), d(w, z)\} > \frac{1}{\lambda^2 \cdot c} \max\{d(u, w), d(y, z)\}$$

Proof. [Proof of Lemma 9] Note that the constraint on not connecting the same pair of chains in the lemma is necessary, because in our sparse spanner construction, it is possible that two points are connected on two different levels on two different pairs of clusters. These two edges could potentially go into different non-empty blocks and get assigned different representatives and cause two parallel edges between two neighborhoods. While this is fine with sparsity purposes as long as there is at most a constant number of such parallel edges, we do not want to have them in our light spanner since they will make the analysis harder. Therefore, we assume that the edges are not connecting clusters centered at the same pair of points.

Next we show that these two pairs are from two cluster levels that are not far from each other. Let (u, w) be an edge on level l of the hierarchy and (y, z) be an edge on level l' of the hierarchy. Without loss of generality, assume that $l \geq l'$. We know that the potential pairs connect same level clusters together. Therefore, the length of (u, w) could vary between R^l and $\lambda \cdot R^l$. A similar inequality holds for (y, z). Thus the ratio of the length of the two would be at least $\lambda^{-1} R^{l-l'}$. Also, if C is chosen large enough it is clear that the two edges must have the same index as well, otherwise the length ratio of C between the two edges would make their endpoints very far from each other. Thus, the edges belong to the same bucket and index, meaning that the length of their ratio is at most c. So,

$$\lambda^{-1} R^{l-l'} < c$$

Now, the separation property on level l' between the clusters that these two edges are connecting to each other states that

$$\max\{d(u, y), d(w, z)\} \geq R^{l'} > \frac{R^l}{\lambda \cdot c}$$

Also according to earlier in this proof, $R^l \geq d(u, w)/\lambda$. Thus,

$$\max\{d(u, y), d(w, z)\} > \frac{d(u, w)}{\lambda^2 \cdot c} = \frac{1}{\lambda^2 \cdot c} \max\{d(u, w), d(y, z)\}$$

□

Now using this lemma we show that the insertion of a potential pair will not cause any violations of Invariant 2 on the same level.

Lemma 10 *Let (u, w) be a potential pair that is inserted to S_i where $i = \text{index}(u, w)$. If $d_i^*(u, w) > (1+\varepsilon')d(u, w)$, then the insertion of (u, w) results in no violations of Invariant 2 on same or lower level edges, assuming that $c^{-1}(1 + \lambda^{-2}) \geq 1 + \varepsilon'$.*

Proof. [Proof of Lemma 10] It is clear that (u, w) cannot participate in a shortest-path (in S_i^*) for any of the lower level pairs, so adding it does not affect any of those pairs. Also adding (u, w) would not violate Invariant 2 for the pair itself because of the assumption $d_i^*(u, w) > (1 + \varepsilon')d(u, w)$. Thus we only need to analyze the other same level edges.

So let (y, z) be a same-level edge in S_i. If one of (u, w) or (y, z) use the other one in its shortest extended path (in S_i^*), then by Lemma 9, the length of the path would be at least

$$\begin{aligned} &\min\{d(u, w), d(y, z)\} + \max\{d(u, y), d(w, z)\} \\ &> \min\{d(u, w), d(y, z)\} + \frac{1}{\lambda^2 \cdot c} \max\{d(u, w), d(y, z)\} \end{aligned}$$

We also know, from the assumption, that (u, w) and (y, z) are same-level edges in S_i, so $c^{-1} < d(u, w)/d(y, z) < c$. Therefore, the stretch of the path would be at least

$$\frac{\min\{d(u,w), d(y,z)\} + \max\{d(u,y), d(w,z)\}}{\max\{d(u,w), d(y,z)\}} > c^{-1}(1 + \lambda^{-2}) \geq 1 + \varepsilon'$$

Thus the stretch of the path is more than $1 + \varepsilon'$, which shows that this addition would not violate Invariant 2 for any of the two pairs, even though the paths of same level edges are excluded in $d_i^*(u, w)$. □

Note that satisfying the condition in Lemma 10 is easy. We first choose large enough λ to have a fine hierarchy, then we choose c small enough that $c < 1 + \lambda^{-2}$, then we choose $\varepsilon' = c^{-1}(1 + \lambda^{-2}) - 1$. Now we show that the potential change on higher level potential pairs would be bounded by a constant after the insertion of (u, w).

Lemma 11 *Let (u, w) be a potential pair that is inserted to S_i where $i = \text{index}(u, w)$. The insertion of (u, w) results in at most*

$$\frac{C_3}{c^k - 1}$$

potential increase on higher level potential pairs in S_i, where

$$C_3 = \varepsilon(1 + \varepsilon)^d c^{d+1} C_1$$

is a constant (and k is the number of buckets).

Proof. [Proof of Lemma 11] Let (y, z) be an edge of level $j' > j$ in S_i whose d_i^* is decreased by the addition of (u, w). Thus the shortest extended path between y and z in S_i^* passes through (u, w). Denote this path by $P_i^*(y, z)$. Before the addition of (u, w), the length of the same path in S_i^* was at most $\|P_i^*(y, z)\| + \varepsilon d(u, w)$. Hence, $\Delta d_i^*(y, z) \geq -\varepsilon d(u, w)$, and the potential change of this edge would be

$$\Delta p_i(y, z) = \frac{-\Delta d_i^*(y, z)}{d(y, z)} \leq \frac{\varepsilon d(u, w)}{d(y, z)} \leq \varepsilon c^{k(j-j')+1}$$

In the next step, we bound the number of such (y, z) pairs. Let r be the minimum length of such edge in level j'. Both y and z must be within $(1 + \varepsilon)cr$ Euclidean distance of u (and w), otherwise the edge (u, w) would be useless in (y, z)'s shortest path in S_i^*. Thus, all such pairs are located in a ball $B(u, (1 + \varepsilon)cr)$, and according to Lemma 2, there would be at most

$$C_2 = (1 + \varepsilon)^d c^d C_1$$

number of them.

Thus, the overall potential change on level j' would be upper bounded by $C_2 \varepsilon c^{k(j-j')+1}$. Summing this up over $j' > j$, the overall potential change on higher level pairs would be at most

$$\Delta\Phi_i < \sum_{j'>j} \varepsilon C_2 c^{k(j-j')+1} = \frac{C_3}{c^k - 1}$$

where $C_3 = \varepsilon C_2 c$. □

Now we analyze the removal of a potential pair from a bucket. The difference with the removal is that it could cause violations of Invariant 1 on its level. Therefore, we analyze a removal, together with some subsequent edge insertions that fix any violations of Invariant 1 on the same level.

Definition 6 (Edge removal process) *Let (u, w) be a potential pair that is located in S_i where $i = \text{index}(u, w)$. We define the single edge removal process on (u, w) to be the process that deletes (u, w) from S_i and fixes the subsequent violations of Invariant 1 on the same level by greedily picking a violating pair, and connecting its endpoints in S_i, until no violating pair for Invariant 1 is left.*

We analyze the effect of the edge removal process in the following two lemmas,

Lemma 12 *Let (u, w) be a potential pair that does not violate Invariant 1 ($d_i^*(u, w) < (1 + \varepsilon)d(u, w)$) and is deleted from S_i ($i = \text{index}(u, w)$), using the edge removal process. The deletion of (u, w) together with these subsequent insertions results in no violations of Invariant 1 or Invariant 2 on same or lower level edges, assuming that $c^{-1}(1 + \lambda^{-2}) \geq 1 + \varepsilon'$.*

Proof. [Proof of Lemma 12] It is clear that (u, w) cannot participate in a shortest-path (in S_i^*) for any of the lower level pairs, so deleting it does not affect any of those pairs. Also, every same level pair that violates Invariant 1 is fixed after the insertion of subsequent edges. Therefore, we just need to show there are no violations of Invariant 2 after these changes. This is also clear by Lemma 10, because we are only inserting edges (y, z) that that violate Invariant 1, i.e. $d_i^*(y, z) > (1 + \varepsilon)d(y, z) > (1+\varepsilon')d(y, z)$, meaning that the assumption of the lemma holds in this insertion. □

We show a similar bound as edge insertion on the effect of the edge removal process on higher level pairs.

Lemma 13 *Let (u, w) be a potential pair that is deleted from to S_i where $i = \text{index}(u, w)$. The edge removal process on (u, w) results in at most*

$$\frac{C_5}{c^k - 1}$$

potential increase on higher level potential pairs in S_i, for some constant C_5 that depends on ε, ε', and c. is a constant.

Proof. [Proof of Lemma 13] The edge removal process can be divided into two phases. The deletion of (u, w), and the insertion of the subsequent pairs. First, we show that the potential increase after the edge deletion is bounded. Let (y, z) be an edge of level $j' > j$ in S_i whose d_i^* is increase by the deletion of (u, w). Thus the shortest extended path between y and z in S_i^* passes through (u, w). Denote this path by $P_i^*(y, z)$. After the removal of (u, w), the length of the same path in S_i^* is at most $\|P_i^*(y, z)\|+\varepsilon d(u, w)$. Hence, $\Delta d_i^*(y, z) \leq \varepsilon d(u, w)$, and the potential change of this edge would be

$$\Delta p_i(y, z) = \frac{\Delta d_i^*(y, z)}{d(y, z)} \leq \frac{\varepsilon d(u, w)}{d(y, z)} \leq \varepsilon c^{k(j-j')+1}$$

Again, the number of such (y, z) pairs is bounded by

$$C_2 = (1 + \varepsilon)^d c^d C_1$$

according to Lemma 2. Thus, the overall potential change on level j' would be upper bounded by $C_2\varepsilon c^{k(j-j')+1}$. Summing this up over $j' > j$, the overall potential change on higher level pairs would be at most

$$\Delta\Phi_i < \sum_{j'>j} \varepsilon C_2 c^{k(j-j')+1} = \frac{C_3}{c^k - 1}$$

where $C_3 = \varepsilon C_2 c$.

Now, the number of subsequent edge insertions would also bounded by a constant. Because in order for an inserted pair (y, z) to violate Invariant 1 after the deletion of (u, w), u and w must be within a distance $c(1+\varepsilon)d(u, w)$, otherwise the edge (u, w) would be useless in their shortest-path. Also since they satisfy Invariant 2, we conclude from Lemma 2 that the number of such pairs is bounded by a constant. Denote this bound by C_4. Then the potential on higher level pairs from the insertions of C_4 pairs on the same level would be at most $C_3C_4/(c^k - 1)$.

Overall, the potential increase on higher level pairs from the edge removal process will be $C_5/(c^k - 1)$ where $C_5 = C_3(C_4 + 1)$. □

Adjusted potential function. We have one last step before analyzing the potential function after a point insertion and a point deletion. We need to slightly adjust the potential function to take into account future edges that might be added between the existing points because of a new point. As we see in section 3, a new point can have a large degree in $\mathcal{S}_0$ due to its implicit clusters in multiple levels of the hierarchy. We handled this by assigning these edges to nearby representatives and we proved a constant degree bound on $\mathcal{S}_1$. But this still would mean adding a point could increase the potential function by $\Omega(\log \Delta)$ since logarithmic number of edges could be added to the sparse spanner. We fix this issue in our potential function by taking into account all the future edges that can be incident to a point. Our adjusted potential function on the whole spanner, denoted by Φ^*, has an extra term compared to the previous potential function Φ,

$$\Phi^* = \Phi + \frac{p_{max}}{2} \cdot \sum_{i=1}^{n} (D_{max} - \deg_{\mathcal{S}_1}(v_i))$$

$\deg_{\mathcal{S}_1}(v_i)$ is the degree of the i-th point (in any fixed order, e.g. insertion order) in the sparse bounded degree spanner $\mathcal{S}_1$, and

$$p_{max} = \max\{1 + \varepsilon, C_\phi(\varepsilon - \varepsilon')\}$$

is the maximum potential value a potential pair can have in its own bucket given the fact that it does not violate Invariant 1. Note that the first term is the maximum of the potential of any pair if its edge is present in the bucket and the second term is the maximum potential of the pair if its edge is absent from the bucket and it is not violating Invariant 1. We will later see why the assumption that Invariant 1 holds for such pairs is fine. But this extra term in the potential function will be used to cover the potential p_i of the extra potential pairs added by the new point.

4.2 Maintaining the light spanner

We are finally ready to introduce our techniques for maintaining a light spanner under a dynamic point set. For point insertion, we select a subset of edges added in the sparse spanner to be present in the light spanner. We show that the potential increase on Φ^* after inserting the new point would be bounded by a constant. Then we perform the same analysis for point deletion and we show that the potential increase is bounded by $\mathcal{O}(\log \Delta)$. In the last part of this section we introduce our methods for iteratively improving the weight of the spanner by showing an algorithm that decreases the potential function by a constant value in each iteration. This concludes our results on the recourse for point insertion and point deletion.

Point insertion. Following a point insertion for a point p, we insert p into the hierarchy and we update our sparse spanner $\mathcal{S}_1$. There are at most a constant number of pairs whose representative assignment has changed, we update these pairs in the light spanner as well. Meaning that if they were present in the light spanner, we keep them present but with the new endpoints, and if they were absent, we keep them absent. Besides the re-assignments, there could be some (even more than a constant) edge insertions into the sparse spanner, but the degree bound of D_{max} would still hold on every point. We greedily pick one new edge at a time that its endpoints violate Invariant 1 in the light spanner, and we add that edge to the light spanner. (Algorithm 3)

Algorithm 3 Inserting a point to the light spanner.

```
1: procedure Insert-to-Light-Spanner(p)
2:     Insert p into the hierarchy T.
3:     Make the required changes on the sparse bounded degree spanner S_1.
4:     for any pair (u, w) with updated representative assignment do
5:         Update the endpoints of the edge in the light spanner.
6:     for any edge (u, w) added to the sparse spanner do
7:         if Invariant 1 is violated for this pair on the light spanner then
8:             Add (u, w) to the light spanner (to its own bucket).
```

Algorithm 4 Deleting a point from the light spanner.

```
1: procedure Delete-from-Light-Spanner(p)
2:     Delete p from the hierarchy T.
3:     Make the required changes on the sparse bounded degree spanner S_1.
4:     for any pair (u, w) removed from the sparse spanner do
5:         Remove (u, w) from the light spanner if present.
6:     for any pair (u, w) added to the sparse spanner do
7:         Add (u, w) to the light spanner.
8:     for any pair (u, w) with updated representative assignment do
9:         Update (u, w) in the light spanner as well.
```

We now analyze the change in the potential function after performing this function following a point insertion.

Lemma 14 Insert-to-Light-Spanner *adds at most a constant amount to* Φ^*.

Proof. [Proof of Lemma 14] Note that at most a constant number of edges will go through a representative assignment change. Each representative change can be divided into removing the old pair and adding the new one. Each removal will increase the potential of at most a constant number of pairs on any same or higher level pairs. This would sum up to a constant amount as we saw earlier in Lemma 12 and Lemma 13. Also, inserting the updated pairs would also sum up to a constant amount of increase in the potential function as we saw in Lemma 10 and Lemma 11.

For the edge insertions however, we will get help from the extra term in our potential function. Note that any extra edge that is added between any two points that existed before the new point will increase both of their degrees by 1 and therefore, decrease the term

$$p_{max} \cdot \sum_{i=1}^{n} (D_{max} - \deg_{\mathcal{S}_1}(v_i))$$

by p_{max}. On the other hand, the new pair will either be added to the light spanner or will satisfy Invariant 1 if not added. Thus, its potential will be at most $1 + \varepsilon$ in the first case, and at most $C_\phi(\varepsilon - \varepsilon')$ in the second case. In any case, the potential of the new pair is not more than p_{max}, and hence Φ^* will not increase due to the addition of the new pair.

Lastly, the new point will introduce a new term $p_{max} \cdot (D_{max} - \deg_{\mathcal{S}_1}(v_{n+1}))$ in Φ^* which would also be bounded by a constant. Overall, the increase in Φ^* will be bounded by a constant. □

Point deletion. Following a point deletion, we perform the deletion on the hierarchy and update the sparse spanner accordingly. This would cause at most $\mathcal{O}(\log \Delta)$ potential pairs to be deleted from or inserted into the spanner. The procedure on the light spanner is simple in this case. We add all the inserted pairs to the light spanner, and we remove the removed pairs from the light spanner if they are present.

Lemma 15 Delete-from-Light-Spanner *adds at most* $\mathcal{O}(\log \Delta)$ *to* Φ^*.

Proof. [Proof of Lemma 15] The number of edges updated on every level of hierarchy after a point removal is bounded by a constant. Therefore, the total number of changes would be bounded by $\mathcal{O}(\log \Delta)$. Each change would cause Φ^* to increase by at most p_{max}. Thus, the total increase is bounded by $\mathcal{O}(\log \Delta)$. □

4.3 Maintenance updates

Our maintenance approach is simple, as long as there exists a potential pair on any S_i that violates either of the two invariants, we perform the corresponding procedure to enforce that invariant for that pair. The fact that the potential function decreases by a constant amount after each fix is the key to our amortized analysis on the number of maintenance updates to reach a spanner with bounded degree and bounded lightness.

Fixing a violation of Invariant 1. In our first lemma in this section, we show that fixing a violation of Invariant 1 in the way that we mentioned above, would decrease the value of the potential function on each S_i.

Lemma 16 *Let* (v, w) *be a potential pair with* $\text{index}(v, w) = i$ *that violates Invariant 1, i.e.* $d_i^*(v, w)/d(v, w) > 1 + \varepsilon$. *Also, assume that*

$$k \geq \log_c \left(1 + \frac{C_3}{(C_\phi - 1)(\varepsilon - \varepsilon')}\right)$$

Then adding the edge (v, w) *to* S_i *decreases the overall potential* Φ_i *of* S_i *by at least* $(\varepsilon - \varepsilon')$.

Proof. [Proof of Lemma 16] Note that adding (v, w) would have no effect on the potential of the lower level or same level potential pairs, due to the definition of d_i^*. We know from Lemma 11 that adding (v, w) to S_i would increase the potential on higher level pairs by at most

$C_3/(c^k-1)$. Also, the potential of the pair itself before the addition is

$$p_i(v,w) = C_\phi \cdot \left(\frac{d_i^*(v,w)}{d(v,w)} - (1+\varepsilon')\right)$$

On the other hand, after the addition,

$$p_i(v,w) = (1+\varepsilon) - \frac{d_i^*(v,w)}{d(v,w)}$$

Therefore,

$$\Delta p_i(v,w) = (\varepsilon - \varepsilon') + (C_\phi + 1)\left(1+\varepsilon' - \frac{d_i^*(v,w)}{d(v,w)}\right)$$

We know by the assumption that the stretch of the shortest extended path between v and w would be more than $1+\varepsilon$, since (v,w) is violating Invariant 1. Therefore,

$$1+\varepsilon' - \frac{d_i^*(v,w)}{d(v,w)} < -(\varepsilon - \varepsilon')$$

Thus,

$$\Delta p_i(v,w) < (\varepsilon - \varepsilon') - (C_\phi+1)(\varepsilon-\varepsilon') = -C_\phi(\varepsilon - \varepsilon')$$

According to this and what we mentioned earlier in the proof,

$$\Delta\Phi_i \le -C_\phi(\varepsilon-\varepsilon') + \frac{C_3}{c^k-1}$$

and if

$$k \ge \log_c\left(1 + \frac{C_3}{(C_\phi-1)(\varepsilon-\varepsilon')}\right)$$

then $\Delta\Phi_i \le -(\varepsilon-\varepsilon')$, which is a negative constant. □

Fixing a violation of Invariant 2. Next, we consider the second type of maintenance updates, which is to fix the violations of Invariant 2. Whenever a pair (v,w) that violates Invariant 2 is found, the first step is to remove the corresponding edge from its subset S_i. Afterwards, we address the same-level violations of Invariant 1 by greedily adding a pair that violates Invariant 1, until none is left. This is the same as performing the edge removal process on the violating pair.

Lemma 17 *Let $(v,w) \in S_i$ be an edge that violates Invariant 2, i.e. $d_i^*(v,w)/d(v,w) \le 1+\varepsilon'$. Also assume that*

$$k \ge \log_c\left(1 + \frac{2C_5}{\varepsilon-\varepsilon'}\right)$$

Then performing the edge removal process on (v,w) decreases the overall potential Φ_i of S_i by at least $(\varepsilon-\varepsilon')$.

Proof. [Proof of Lemma 17] Since all the additions and removals in the edge removal process are happening on the same level and also due to the definition of d_i^*, there would be no potential change on any of the same or lower level pairs. We know from Lemma 13 that deleting (v,w) from S_i would increase the potential on higher level pairs by at most $C_5/(c^k-1)$. The potential of the pair itself before the deletion is

$$p_i(v,w) = (1+\varepsilon) - \frac{d_i^*(v,w)}{d(v,w)}$$

After the deletion,

$$p_i(v,w) = C_\phi \cdot \left(\frac{d_i^*(v,w)}{d(v,w)} - (1+\varepsilon')\right)$$

Therefore,

$$\Delta p_i(v,w) = -(\varepsilon - \varepsilon') - (C_\phi + 1)\left(1+\varepsilon' - \frac{d_i^*(v,w)}{d(v,w)}\right)$$

We know by the assumption that the stretch of the shortest extended path between v and w would be less than $1+\varepsilon'$, since (v,w) is violating Invariant 2. Therefore,

$$1+\varepsilon' - \frac{d_i^*(v,w)}{d(v,w)} > 0$$

Thus,

$$\Delta p_i(v,w) < (\varepsilon - \varepsilon')$$

According to this and what we mentioned earlier in the proof,

$$\Delta\Phi_i \le -(\varepsilon-\varepsilon') + \frac{C_5}{c^k-1}$$

and if

$$k \ge \log_c\left(1 + \frac{2C_5}{\varepsilon-\varepsilon'}\right)$$

then $\Delta\Phi_i \le -(\varepsilon-\varepsilon')/2$, which is a negative constant. □

Bounding the number of updates. Now that we introduced our maintenance updates and we analyzed the change in the potential functions after each of these updates, we can finally prove our amortized bounds. We prove that the amortized number of edge updates in our algorithm after a point insertion is $\mathcal{O}(1)$, while the amortized number of edge updates after a point deletion is $\mathcal{O}(\log\Delta)$.

Theorem 18 *Our fully-dynamic spanner construction in d-dimensional Euclidean spaces has a stretch-factor of $1+\varepsilon$ and a lightness that is bounded by a constant. Furthermore, this construction performs an amortized $\mathcal{O}(1)$ edge updates following a point insertion, and an amortized $\mathcal{O}(\log\Delta)$ edge updates following a point deletion.*

Proof. [Proof of Theorem 18] The stretch factor and the lightness immediately follow from the fact that our spanner always satisfies the two invariants, and according to Lemma 1 and the leapfrog property, that would be enough for a $1+\varepsilon$ stretch factor and constant lightness.

In order to prove the amortized bounds on the number of edge updates after each operation, we recall that by Lemma 14, the potential change $\Delta\Phi^*$ after a point insertion is bounded by a constant, and by Lemma 15, the potential change after a point deletion is bounded by $\mathcal{O}(\log \Delta)$. On the other hand, by Lemma 16 and Lemma 17, each maintenance update reduces the potential Φ^* by at least $(\varepsilon - \varepsilon')/2$, since the impacted Φ_i reduces after the maintenance update, Φ_j for $j \neq i$ will remain unchanged, and the extra term $\frac{p_{max}}{2} \cdot \sum_{i=1}^{n}(D_{max} - \deg_{\mathcal{S}_1}(v_i))$ will also remain unchanged since the sparse spanner is not affected by the maintenance updates. Therefore, the amortized number of maintenance updates required after each point insertion is $\mathcal{O}(1)$ while this number after a point deletion is $\mathcal{O}(\log \Delta)$. Also, the number of edge updates before the maintenance updates would be bounded by the same amortized bounds. Thus, we can finally conclude that the amortized number of edge updates following a point insertion is $\mathcal{O}(1)$, while for a point deletion it is $\mathcal{O}(\log \Delta)$. □

References

[1] 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.

[2] S. Arya, D. M. Mount, and M. H. M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20–22 November 1994*, pages 703–712. IEEE Computer Society, 1994.

[3] A. S. Biswas, M. Dory, M. Ghaffari, S. Mitrović, and Y. Nazari. Massively parallel algorithms for distance approximation and spanners. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 118–128, 2021.

[4] G. Borradaile, H. Le, and C. Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6–9, 2019*, pages 2371–2379. SIAM, 2019.

[5] P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry*, 28(1):11–18, 2004.

[6] R. Cole and L. Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In J. M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21–23, 2006*, pages 574–583. ACM, 2006.

[7] G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished euclidean graphs. In *proceedings of the sixth annual ACM-SIAM symposium on discrete algorithms*, pages 215–222, 1995.

[8] D. Eppstein and H. Khodabandeh. On the edge crossings of the greedy spanner. In K. Buchin and É. C. de Verdière, editors, *37th International Symposium on Computational Geometry, SoCG 2021, June 7–11, 2021, Buffalo, NY, USA (Virtual Conference)*, volume 189 of *LIPIcs*, pages 33:1–33:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[9] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Computational Geometry*, 35(1-2):2–19, 2006.

[10] L.-A. Gottlieb and L. Roditty. An optimal dynamic spanner for doubling metric spaces. In D. Halperin and K. Mehlhorn, editors, *Algorithms – ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15–17, 2008, Proceedings*, volume 5193 of *Lecture Notes in Computer Science*, pages 478–489. Springer, 2008.

[11] L.-A. Gottlieb and L. Roditty. Improved algorithms for fully dynamic geometric spanners and geometric routing. In S.-H. Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20–22, 2008*, pages 591–600. SIAM, 2008.

[12] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 150–158, 2005.

[13] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In J. I. Munro, editor, *Proceedings of the Fifteenth Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11–14, 2004*, pages 798–807. SIAM, 2004.

[14] H. Le and C. Than. Greedy spanners in Euclidean spaces admit sublinear separators. In J. S. Naor and N. Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3287–3310. SIAM, 2022.

[15] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[16] L. Roditty. Fully dynamic geometric spanners. *Algorithmica*, 62(3):1073–1087, 2012.

[17] O. Salzman, D. Shaharabani, P. K. Agarwal, and D. Halperin. Sparsification of motion-planning roadmaps by edge contraction. *The International Journal of Robotics Research*, 33(14):1711–1725, 2014.

On Totally-Concave Polyominoes*

Gill Barequet† Noga Keren† Johann Peters‡ Neal Madras§ Adi Rivkin†

Abstract

A polyomino is an edge-connected set of cells on the square lattice. Every row or column of a *totally-concave* (TC) polyomino consists of more than one sequence of consecutive cells of the polyomino. We show that the minimum area (number of cells) of a TC polyomino is 21 cells. We also suggest, implement, and run an efficient algorithm for counting TC polyominoes. Finally, we prove that the associated sequence $(\kappa(n))$ has a finite growth constant λ_κ, prove the lower bound $\lambda_\kappa > 2.4474$, and conjecture that λ_κ is equal to the growth constant of *all* polyominoes.

1 Introduction

A *polyomino* of area n is a connected set of n cells on the square lattice $\mathbb{Z}^2$, where connectivity is through edges. Two polyominoes are considered equivalent if one can be transformed into the other by a translation.

Counting polyominoes is a long-standing problem in discrete geometry, originating in statistical physics in the context of percolation processes [10] and popularized in Golomb's pioneering book [12] and by M. Gardner's columns in *Scientific American*. The sequence $A(n)$, which lists the number of polyominoes, is currently known up to $n = 70$ [1].

The growth constant of polyominoes has also attracted much attention in the literature. Klarner [16] showed that the limit (a.k.a. *Klarner's constant*) $\lambda := \lim_{n\to\infty} \sqrt[n]{A(n)}$ exists. The convergence of $A(n+1)/A(n)$ to λ, as $n \to \infty$, was proved only three decades later by Madras [17]. The best-known lower [4] and upper [5] bounds on λ are 4.0025 and 4.5252, respectively. By applying numerical methods to the known values of $A(n)$, it is widely believed that $\lambda \approx 4.06$, and the currently best estimate of λ is 4.0625696 ± 0.0000005 [15]. (Based on the new counts of $A(n)$ till $n = 70$, a better estimate, 4.06256912(2), was provided to us by I. Jensen in a personal communication.)

*Work on this paper by the fourth author has been supported in part by a Discovery Grant from NSERC Canada.

†Dept. of Computer Science, Techion—Israel Inst. of Technology, Haifa 3200003, Israel, barequet@cs.technion.ac.il, [noga.keren,adi.rivkin]@campus.technion.ac.il

‡Dept. of Mathematics, Univ. of Waterloo, ON N2L 3G1, Canada, j8peters@uwaterloo.ca

§Dept. of Mathematics and Statistics, York University, Toronto, ON M3J 1P3, Canada, madras@yorku.ca.

In a *convex* polyomino, each row and column consists of exactly one maximal continuous sequence of cells. These polyominoes are essential in many application domains, and they attracted a considerable amount of attention in the literature. See, for example, a discussion of the asymptotic number of convex polyominoes [8], a derivation of a rather complex generating function for the sequence that enumerates convex polyominoes [9], a method for generating random convex polyominoes [13], and an investigation of the relation between ordering and convex polyominoes [11], among many other works.

However, the complement type of polyominoes was hardly considered. In a *totally-concave* (TC) polyomino, each row and column consists of at least *two* maximal continuous sequences of cells, as is shown in Figure 1.[1] It is hinted in Ref. [7, §14, p. 369, problem 14.5.4] that the minimum possible area of a TC polyomino is 21. Let $\kappa(n)$ be the number of TC polyominoes of size (area) n. An algorithm for computing $\kappa(n)$, for a given n, is also sought as an open problem [7, §14, p. 369, problem 14.5.5]. Among other results, we settle the minimality conjecture and suggest an efficient algorithm.

The paper is organized as follows. In Section 2, we prove that there do not exist TC polyominoes of area less than 21. In Section 3, we present a nontrivial extension of Jensen's algorithm to counting TC polyominoes, and report counts of these polyominoes up to size 35. In Section 4, we prove that the sequence $\kappa(n)$ has a growth constant λ_κ, prove that $\lambda_\kappa > 2.4474$, and provide a motivation for the conjecture that $\lambda_\kappa = \lambda$. We end in Section 5 with some concluding remarks and future research directions.

2 Minimum Area

Theorem 1 *The minimum area of a TC polyomino is 21.*

The proof of this theorem follows a necessity-sufficiency format. Necessity is shown by deducing upper and lower bounds on the area of TC polyominoes in $m \times \ell$ bounding boxes; These bounds contradict each other for areas less than 21. Sufficiency is evident by example.

[1] Recipe for the picture in Figure 1(a.2) is available upon request.

(a.1) Symbolic (a.2) Eatable (a) Area 21

With marked edges (b) Area 24

Figure 1: TC polyominoes of various areas and flavors. The symbolic representation in (b) distinguishes between *hidden* edges (green), *inside* edges (blue), and *outside* edges (red).

Proof. A lower bound on the area of a TC polyomino within an $m \times \ell$ bounding box is achieved by partitioning the edges of such a polyomino into *hidden*, *outside*, and *inside* edges, as shown in Figure 1(b). The top (resp., right/bottom/left) edge of a cell c is *hidden* if there is a cell of the polyomino immediately above (resp., to the right of/below/to the left of) c. An edge is *outside* if it is not facing any other edge. An *inside* edge is an edge facing another edge, but not immediately, that is, with a gap of at least one cell. Consider a TC polyomino. Denote by n its area, and by i, o, and h the number of inside, outside, and hidden edges, respectively, of the polyomino. For example, by these definitions, the "U-pentomino" ( ) has $i = 2$, $o = 10$, and $h = 8$. For the area-24 TC-polyomino depicted in Figure 1(b), we have $i = 26$, $o = 24$, and $h = 46$. By pairing inside and outside edges in rows and columns, we have that $o = 2m + 2\ell$ and $i \geq 2m + 2\ell$. We also have that $h \geq 2n - 2$ since the polyomino is connected and, hence, it must include at least $n-1$ cell adjacencies. Since $h + o + i = 4n$, we have that $n \geq 2m + 2\ell - 1$.

For an upper bound on n, we may assume without loss of generality that $m \leq \ell$. Then, a TC polyomino within an $m \times \ell$ bounding box must be missing at least one cell from each of the ℓ columns, none of which is in the top or bottom row (for guaranteeing concavity of the columns), as well as at least two further cells, one in the top and one in the bottom row (for guaranteeing concavity of these rows). Therefore, $n \leq m\ell - \ell - 2$.

Altogether, we have that $2m+2\ell-1 \leq n \leq m\ell-\ell-2$, with $m \leq \ell$. A simple case analysis shows that the smallest n satisfying these constraints is 21, with $m = 5$ and $\ell = 6$.

Hence, $n \geq 21$ is a necessary condition for a TC polyomino. On the other hand, the existence of a TC polyomino of area 21 is evident by Fig. 1(a). This completes the proof. □

This result was confirmed by our TC-polyomino counting programs (see Section 3). Figure 2 shows representatives of the 152 TC polyominoes of area 21.

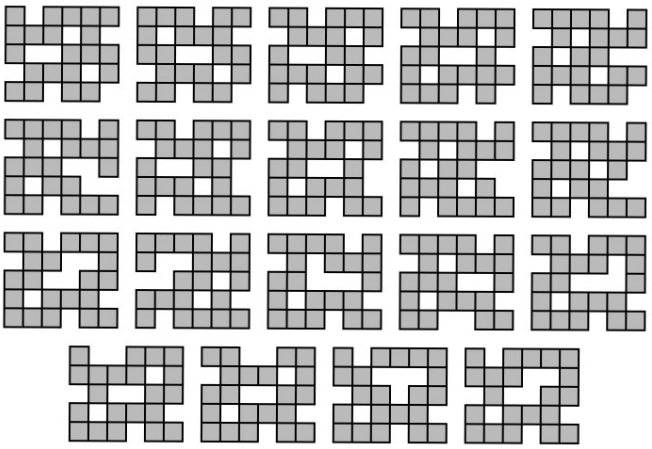


Figure 2: The 19 TC polyominoes of area 21, up to rotation and mirroring.

(None of these polyominoes have any symmetries, hence, the polyominoes formed by the eight orientations of each of the 19 drawn polyominoes are distinct.)

3 An Efficient Counting Algorithm

3.1 Algorithm

We first implemented a prototype backtracking algorithm for counting TC polyominoes. The program recursively concatenated concave columns to a growing polyomino. A branch of this procedure was abandoned when the area of the polyomino grew too large or if it was no longer possible for it to become connected with the addition of further columns. (This happened when a component of the polyomino became permanently detached.)

We then designed a much more efficient algorithm, based on Jensen's algorithm for counting all polyominoes [14, 15]. In a nutshell, Jensen's algorithm counts polyominoes within horizontal bounding strips of height h, where $1 \leq h \leq \lceil n/2 \rceil$. The algorithm con-

siders column by column from left to right, and cell by cell from top to bottom within each column. At each cell, the algorithm considers either to have it occupied (belonging to the polyomino) or empty (not belonging). At all stages, the algorithm does not keep in memory all polyominoes but all possible *right boundaries* of polyominoes, that is, all combinations of the last h cells considered. The algorithm maintains a database whose entries have keys that are the different *signatures*, where a signature consists of a boundary plus all possible connections between cells on the boundary by cells found to the left of it. In other words, the keys reflect all possible splits of boundary cells into connected components, where the connections are to the left of the boundary. In addition, a signature also includes two bits that indicate whether or not the polyominoes associated with that entry touch the top and/or bottom of the strip. The contents of each entry in the database is statistics of all partially-built polyominoes ("partially" means that polyominoes may still consist of more than one connected component), that is, the *counts* of all polyominoes parameterized by area, having that specific signature. When the currently considered cell is chosen to be occupied, the counts of polyominoes are updated by adding the numbers of fully-built polyominoes, that is, polyominoes that consist of exactly one connected component and touch the top and bottom of the strip.

Our modifications. For counting TC polyominoes, we also need to ensure that each column and each row consists of more than one consecutive sequence of cells. This is simple to achieve for columns: At the end of processing a column, we discard from the database all entries that correspond to columns that contain less than two sequences of occupied cells. For rows, we enhance the signatures by splitting each one into at most 4^h subsignatures: For each row, we keep a code as follows: '0' indicates that the first sequence of occupied cells has not been met yet; '1' means that we are in the middle of the first sequence; '2' states that we are between the first and second sequences; and '3' signifies that we have already entered the second sequence. (Once we reach '3,' we do not need to update this indicator any more.) Figure 3 shows an enhanced signature, in which each boundary cell is associated with two numbers: The original vertical code (left), and the additional horizontal code (right). Then, we count only polyominoes with signatures whose line indicators are all '3.' Note that the indicators of the top and bottom rows make the two bits described above redundant.

Jensen's algorithm is efficient in the sense that it is the only known algorithm whose running time, $\tilde{O}(1.732^n)$ [3], is smaller than the total number of polyominoes, $\tilde{\Theta}(\lambda^n)$. (Recall that $\lambda \approx 4.063$.) Our modification splits every signature into at most $4^{n/2} = 2^n$ subsignatures (in practice, into much less than that), thus, the running time of the modified algorithm is $\tilde{O}(3.464^n)$, which is still much smaller than the total number of polyominoees. In conclusion, our version of the algorithm is slower than the original algorithm, although we eventually count fewer polyominoes, due to the exponential growth in the number of processed signatures.

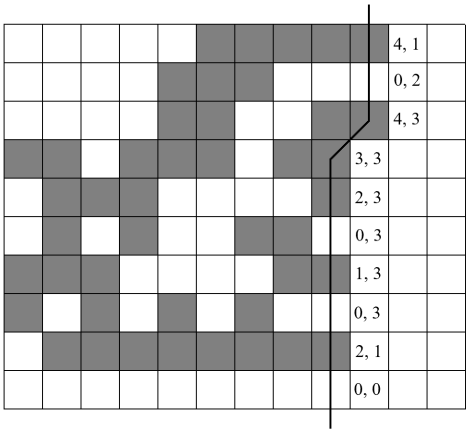


Figure 3: An enhanced boundary signature in the modified version of Jensen's algorithm.

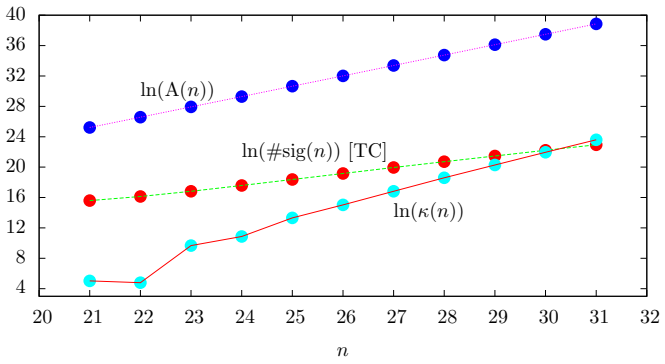


Figure 4: Plots of the number of signatures (while counting TC polyominoes), all poyominoes, and TC polyominoes.

Figure 4 plots in a semi-logarithmic scale the number of distinct signatures encountered by the algorithm while computing $\kappa(n)$) (in red circles), together with the number of TC polyominoes (cyan) and the total number of polyominoes (blue), all as functions of n, for $21 \le n \le 31$.

3.2 Results

Our prototype program, implemented in Python, computed in 90 hours (elapsed time) $\kappa(n)$ up to $n = 26$ on a PC with a 64-bit system operating an i5-9400F Intel Core CPU at 2.90GHz with 12GB of RAM.

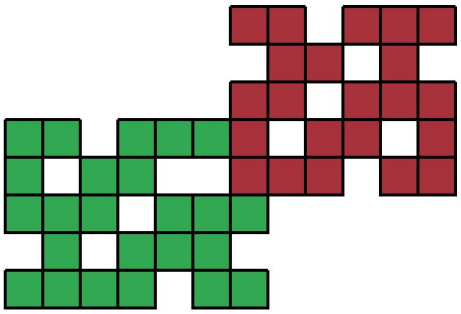


Figure 5: The concatenation of two TC polyominoes is always a TC polyomino.

The modified version of Jensen's algorithm was implemented in `C++` and run on a 12th generation Intel(R) i9-12900KF with 128GB of RAM. Using about 41 hours of CPU, the program computed $\kappa(n)$ up to $n = 35$, obtaining the values reported in Table 1 and agreeing with all values computed by the prototype program.

4 Growth Constant

Bender [8] showed that the number of convex polyominoes of size n is asymptotically $t\gamma^n$, for $\gamma \approx 2.3091$ and $t \approx 2.6756$, that is, the growth constant (see a formal definition below) of convex polyominoes is roughly 2.3091. In this section, we investigate the growth constant of TC polyominoes.

4.1 Existence

Definition 1 *(lexicographic order) For cells c_1, c_2, we say that $c_1 \prec c_2$ if c_1 lies in a column which is to the left of the column of c_2,* or *if c_1 lies below c_2 in the same column.*

Definition 2 *(concatenation) Let P_1, P_2 be two polyominoes, and let c_1 (resp., c_2) be the largest (resp., smallest) cell of P_1 (resp., P_2). The* concatenation *of P_1 and P_2 is the placement of P_2 relative to P_1, such that c_2 is found immediately on top of c_1.*

Figure 5 shows the concatenation of two polyominoes P_1 and P_2. The result of concatenating P_1 and P_2 is always a valid polyomino since the two polyominoes touch each other but do not overlap. Moreover, if both P_1 and P_2 are TC, then the result of concatenating them is also TC.

Theorem 2 *The limit $\lambda_\kappa := \lim_{n\to\infty} \sqrt[n]{\kappa(n)}$ (the growth constant of $(\kappa(n))$) exists and is finite.*

Proof. We follow the proof of existence and finiteness of Klarner's constant λ [16]. First, the sequence $\kappa(n)$ is supermultiplicative, that is, $\kappa(n)\kappa(m) \leq \kappa(n+m)$ for all $m, n \in \mathbb{N}$. This is justified by a simple concatenation argument. Indeed, all TC polyominoes of area n can be concatenated with all TC polyominoes of area m (see, *e.g.*, Figure 5), yielding distinct TC polyominoes of area $n+m$. Second, there exists a constant $\mu > 0$ for which $\kappa(n) \leq \mu^n$ for all $n \in \mathbb{N}$. For example, $\mu = \lambda$, the growth constant of all polyominoes. (This follows immediately from the fact that $\kappa(n) \leq A(n) \leq \lambda^n$.) By a lemma of Fekete (Klarner cites Ref. [18, p. 852] for similar results), the claim follows. □

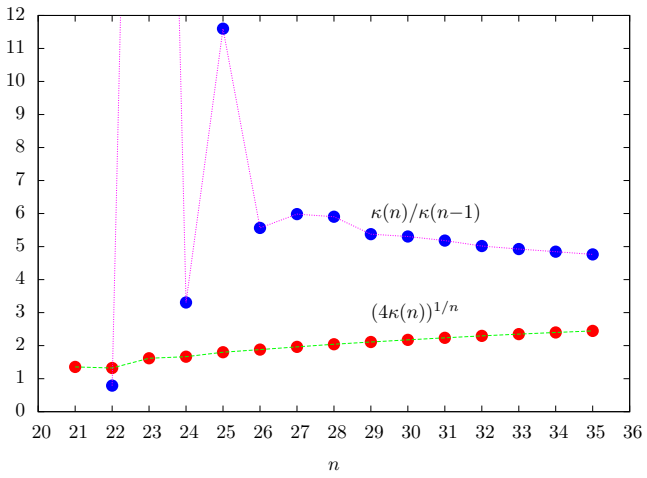


Figure 6: Plots of known values of $(4\kappa(n))^{1/n}$ and $\kappa(n)/\kappa(n-1)$.

It would be much more ambitious to prove the existence of the ratio sequence, that is, $\lim_{n\to\infty} \frac{\kappa(n)}{\kappa(n-1)}$. Obviously, if it exists, it must be equal to λ_κ.

Remark In fact, it makes more sense (see Section 4.2) to explore $((4\kappa(n))^{1/n})$ instead of $((\kappa(n))^{1/n})$. Figure 6 shows plots of the known values of $(4\kappa(n))^{1/n}$ and $\kappa(n)/\kappa(n-1)$. Surprisingly, the ratio sequence seems empirically to be monotone *decreasing* (except some low-order fluctuations), a property rarely found in other families of polyominoes.

4.2 Lower Bound

We now present a computer-assisted proof of a lower bound on λ_κ.

Definition 3 *(composition) A* composition *of two polyominoes is a relative placement of the two polyominoes, such that they touch (edge to edge), possibly in multiple places, but do not overlap.*

Figure 7 shows a few compositions of a pair of polyominoes P, Q. Some compositions (*e.g.*, those shown in Figures 7(b-d)) are *lexicographic*, that is, compositions in which all cells of P are lexicographically smaller than

Table 1: Counts of TC polyominoes.

n	$\kappa(n)$	n	$\kappa(n)$	n	$\kappa(n)$	n	$\kappa(n)$
1–20	0	24	52,306	28	119,309,768	32	88,476,873,440
21	152	25	606,636	29	641,447,812	33	435,921,253,072
22	120	26	3,376,528	30	3,403,173,276	34	2,113,011,155,472
23	15,820	27	20,204,672	31	17,634,751,456	35	10,065,872,407,536

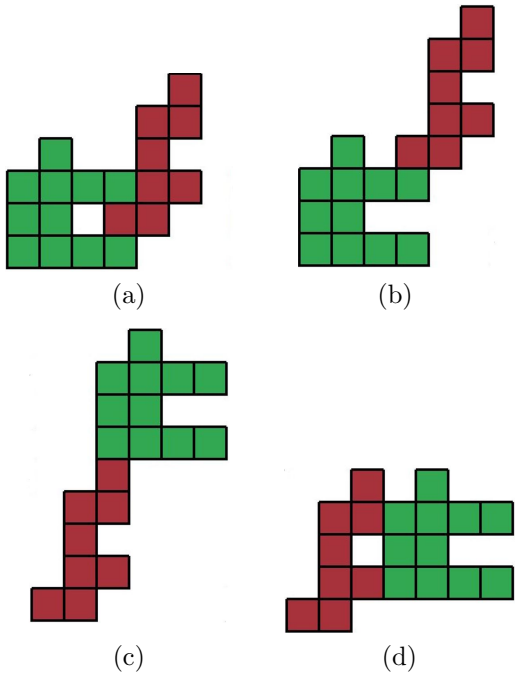


Figure 7: A few compositions of a sample pair of polyominoes.

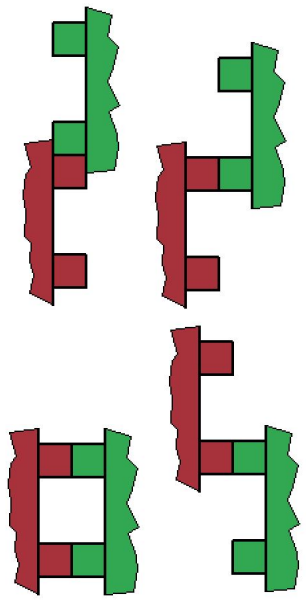


Figure 8: There are at least four lexicographic compositions of any pair of TC polyominoes.

all cells of Q (or vice versa), while other compositions (see, *e.g.*, Figure 7(a)) are not lexicographic. It is easy to observe that a composition of two TC polyominoes is not always a TC polyomino. However, any *lexicographic* composition of two TC polyominoes is also TC.

Lemma 3 *(A simplified version of Theorem 1(a) in Ref. [2, p. 3]) Assume that the limit* $\mu := \lim_{n\to\infty} \sqrt[n]{Z(n)}$ *exists for a sequence* $(Z(n))$*. Let* $c_1 > 0, c_2$ *be some constants. Then, if* $c_1 n^{c_2} Z^2(n) \leq Z(2n)\ \forall n \in \mathbb{N}$*, then* $\sqrt[n]{c_1 (2n)^{c_2} Z(n)} \leq \mu\ \forall n \in \mathbb{N}$.

Theorem 4 $\lambda_\kappa > 2.4474$.

Proof. We use a composition argument, using the property that the extreme (rightmost and leftmost) columns of any TC polyomino have at least two cells. This property allows at least four lexicographic compositions of any pair of TC polyominoes P, Q that yield TC polyominoes. It can easily be verified that the minimum number of such compositions is obtained when both the rightmost column of P and the leftmost column of Q contain exactly two cells, with the same vertical gap between them. For such pairs of TC polyominoes, we have the four lexicographic compositions shown in Figure 8. Indeed, if the gaps between these cells are different (as seen in Figure 9), the two TC polyominoes P, Q have five lexicographic compositions; and if the respective columns of P, Q have more than two occupied cells, the number of lexicographic compositions may only increase.

Consequently, we have that $4(\kappa(n))^2 \leq \kappa(2n)$. Then, Lemma 3 implies that any term of the form $(4\kappa(n))^{1/n}$ is a lower bound on λ_κ. Checking the known values of $\kappa(n)$, we see that $n = 35$ provides the best lower bound $\lambda_\kappa \geq (4\kappa(35))^{1/35} > 2.4474$. □

4.3 Conjectured Value

Figure 4 may suggest that the growth constant of TC polyominoes is identical to that of all polyominoes. We state this as a conjecture and provide for it a tentative proof that depends on another well-known conjecture about the average diameter of lattice animals.

Conjecture 1 $\lambda_\kappa = \lambda$.

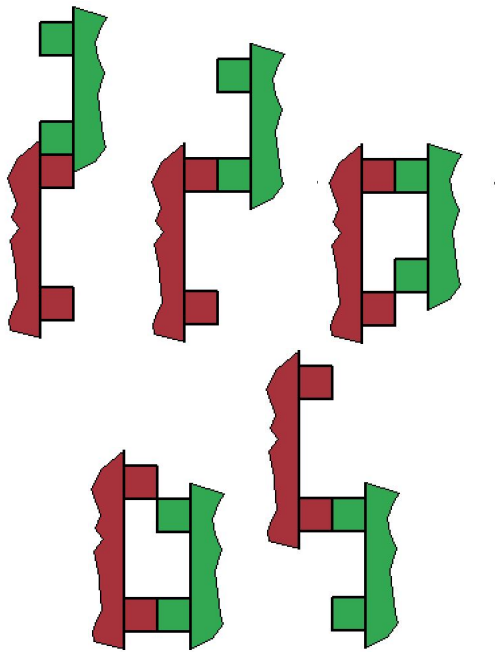


Figure 9: Five lexicographic compositions of a pair of TC polyominoes.

It is generally *believed* [19, §9.2] that all standard models of lattice animals and lattice trees (including polyominoes) have average diameter that scales as n^ν for some critical exponent $\nu < 1$ that depends only on the dimension of the lattice. Numerically, $\nu \approx 0.64$ in two dimensions. This is borne out in several numerical and theoretical studies in the physics literature. Here we can define the "diameter" of a polyomino P as the maximum Euclidean distance between any two cells of P. In particular, since (it is believed that, say) $\nu < 0.9$, let U_n be the set of all polyominoes of size n whose diameter is less than $n^{0.9}$. Then, the above belief implies that $|U_n| > A(n)/2$ for all sufficiently-large n.

Refer to Figure 10. Let L be the L-shaped frame depicted in red in the figure. Its width and height are $n^{0.9}$. Let $\alpha(n)$ be the number of cells in L. Then, $\alpha(n) = \Theta(n^{0.9})$. For any polyomino $P \in U_n$, let $f(P)$ be the union of L with the translation of P (colored in green) that has the lower left corner of its bounding box at (0,0). Then, $f(P)$ is a TC polyomino, and its area is $n+\alpha(n)$. Since the function $f(\cdot)$ is clearly one-to-one, we deduce that $\kappa(n + \alpha(n)) \geq |U_n|$. It follows that

$$\lambda_\kappa^{n+\alpha(n)} \geq \kappa(n + \alpha(n)) \geq A(n)/2$$

for all sufficiently-large n. Now take nth roots of the above, and let $n \to \infty$. The leftmost side converges to λ_κ, and the rightmost side converges to λ. We conclude that $\lambda_\kappa \geq \lambda$. The reverse relation is trivial, hence, $\lambda_\kappa = \lambda \approx 4.06$.

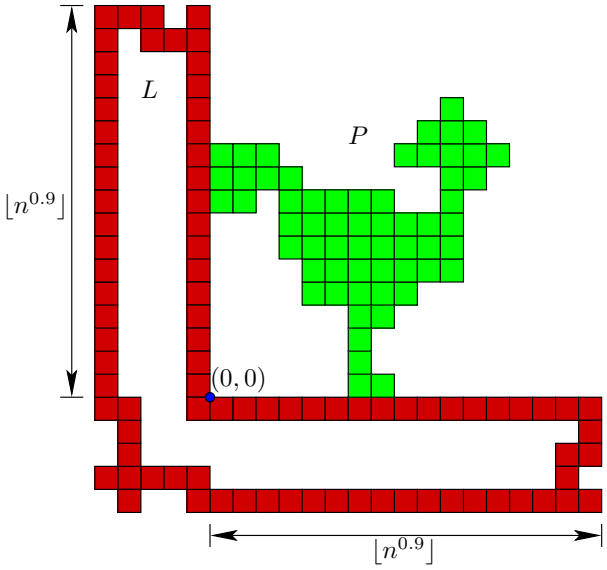


Figure 10: The function $f(P)$.

To the best of our knowledge, if this conjecture were true, then the family of TC polyominoes would be the only nontrivial proper subset of polyominoes previously studied in the literature that has been shown to have the same growth constant as all polyominoes.

5 Conclusion and Future Work

In this paper, we investigate a few problems related to TC polyominoes. We prove that the minimum possible area of such a polyomino is 21; suggest an efficient algorithm for counting TC polyominoes, and report counts of TC polyominoes till area 35; show that $(\kappa(n))$, the sequence of counts of TC polyominoes of area n, has a growth constant λ_κ; prove that $\lambda_\kappa > 2.4474$; and finally, conjecture that $\lambda_\kappa = \lambda \approx 4.06$.

Our main future research directions are the following.

1. Prove the existence of the limit of the ratio sequence, that is, $\lim_{n\to\infty} \frac{\kappa(n)}{\kappa(n-1)}$. (As noted above, if the limit exists, then it must be equal to λ_κ.)
2. Set a good *upper* bound on λ_κ. (Traditionally, upper bounds are harder to obtain than lower bounds).

Other future research directions include a few subfamilies of TC polyominoes.

Definition 4 *(minimality) A TC polyomino P is* minimal *if no proper subset of cells of P is TC.*

Duplicating any row or column of a TC polyomino results in a TC polyomino. The opposite is also true: Discarding all but at least one of consecutive identical rows or columns of a TC polyomino results in a TC polyomino. This gives rise to the following definition.

Definition 5 *(primitivity) A TC polyomino is* primitive *if it does not contain any consecutive identical rows or columns.*

It is also worth considering TC polyominoes whose bounding boxes are "full."

Definition 6 *(saturation) A TC polyomino P is* saturated *if no empty cells in the bounding box of P can be filled and added to P, such that the result is still a TC polyomino.*

Here are some more questions to explore.

3. Are there members of the above subfamilies of unlimited size? (We found minimal, primitive, and saturated TC polyominoes of unlimited size.)
4. Is the intersection between the above subfamilies non-empty?
5. Do the sequences that enumerate the above subfamilies have growth constants? (For these subfamilies, we cannot apply concatenation arguments since the concatenation of pairs of minimal or saturated TC polyominoes always result in polyominoes which do not belong to these subfamilies, and the concatenation of pairs of primitive TC polyominoes might result in TC polyominoes which are not primitive.)
6. Design efficient algorithms for counting members of the above subfamilies. (At a first glance, it seems that extending Jensen's algorithm for any of the above subfamilies is unlikely since the properties defining the subfamilies are global.)

Further research directions involve more general settings of the problem.

7. Consider polyominoes in which each row and column contains at least $k > 2$ (say, 3) maximal sequences of occupied cells.
8. Explore similar problems in other planar lattices (*e.g.*, the triangular or hexagonal lattice).
9. Investigate similar problems for polycubes (face-connected sets of cells on cubical lattices) in higher dimensions. (Note the two possible *different* definitions of total concavity in a higher dimension d: A "weak" total concavity would require that every line parallel to one of the coordinate axes cross the polycube in either 0 or at least two maximal sequences of cells; A "strong" total concavity would require recursively (for $d > 2$) that the intersection of every $(d-1)$-dimenisonal hyperplane, perpendicular to one of the coordinate axes, be either empty or a $(d-1)$-dimenisonal TC polycube, where total concavity in two dimensions is as defined in this paper.)

References

[1] G. Barequet and G. Ben-Shachar, Counting polyominoes, revisited, *SIAM Symp. on Algorithm Engineering and Experiments*, Alexandria, VA, pp. 133–143, January 2024.

[2] G. Barequet, G. Ben-Shachar, and M.C. Osegueda, Concatenation arguments and their applications to polyominoes and polycubes, *Computational Geometry: Theory and Applications*, 98 (2021), 12 pp.

[3] G. Barequet and M. Moffie, On the complexity of Jensen's algorithm for counting fixed polyominoes, *J. of Discrete Algorithms*, 5 (2007), 348–355.

[4] G. Barequet, G. Rote, and M. Shalah, $\lambda > 4$: An improved lower bound on the growth constant of polyominoes, *Comm. of the ACM*, 59 (2016), 88–95.

[5] G. Barequet and M. Shalah, Improved upper bounds on the growth constants of polyominoes and polycubes, *Algorithmica*, 84 (2022), 3559–3586.

[6] G. Barequet, M. Shalah, and Y. Zheng, An improved lower bound on the growth constant of polyiamonds, *J. of Combinatorial Optimization*, 37 (2019), 424–438.

[7] G. Barequet, S.W. Solomon, and D.A. Klarner, Polyominoes, *Handbook of Discrete and Computational Geometry*, 3rd ed. (E. Goodman, J. O'Rourke, and C.D. Tóth, eds.), 359–380. Chapman and Hall/CRC Press, 2017.

[8] E.A. Bender, Convex n-ominoes, *Discrete Mathematics*, 8 (1974), 219–226.

[9] M. Bousquet-Mélou and J.M. Fédou, The generating function of convex polyominoes: The resolution of a q-differential system, *Discrete Mathematics*, 137 (1995), 53–75.

[10] S.R. Broadbent and J.M. Hammersley, Percolation processes: I. Crystals and mazes, *Proc. Cambridge Philosophical Society*, 53 (1957), 629–641.

[11] G. Castiglione and A. Restivo, Ordering and convex polyominoes, *Proc. 4th Int. Conf. on Machines, Computations, and Universality*, Saint Petersburg, Russia, September, 128–139, 2004 (Revised Selected Papers 4, Springer Berlin Heidelberg, 2005).

[12] S.W. Golomb, *Polyominoes*, Princeton University Press, Princeton, NJ, 1965 (2nd ed., 1994).

[13] W. Hochstättler, M. Loebl, and C. Moll, Generating convex polyominoes at random, *Discrete Mathematics* 153 (1996), 165–176.

[14] I. Jensen, Enumerations of lattice animals and trees, *J. of Statistical Physics*, 102 (2001), 865–881.

[15] I. Jensen, Counting polyominoes: A parallel implementation for cluster computing, *Proc. Int. Conf. on Computational Science*, part III, Melbourne, Australia and St. Petersburg, Russia, *Lecture Notes in Computer Science*, 2659, Springer, 203–212, June 2003.

[16] D.A. Klarner, Cell growth problems, *Canadian J. of Mathematics*, 19 (1967), 851–863.

[17] N. Madras, A pattern theorem for lattice clusters, *Annals of Combinatorics*, 3 (1999), 357–384.

[18] G. Pólya and G. Szegő, Aufgaben und Lehrsätze aus der Analysis Bd. Funktionentheorie. Nullstellen. Polynome. Determinanten. Zahlentheorie (Tasks and theorems from analysis: Vol. on Function theory, zeros, polynomials, determinants, number theory), vol. 2, Springer, 1925.

[19] C. Vanderzande, *Lattice models of polymers*, no. 11, Cambridge University Press, 1998.

Skeletal Cut Loci on Convex Polyhedra*

Joseph O'Rourke Costin Vîlcu

Abstract

For a point x on a convex polyhedron P, the *cut locus* $C(x)$ is the closure of the set of points on P joined to x by at least two geodesic segments (shortest paths) on P. It forms a tree of geodesic segments that includes every vertex of P. We say that P has a *skeletal cut locus* if there is some $x \in P$ such that $\mathcal{C}(x) \subset \mathrm{Sk}(P)$, where $\mathrm{Sk}(P)$ is the 1-*skeleton* of P. At a first glance, there seems to be very little relation between the cut locus and the 1-skeleton, as the first one is an intrinsic geometry notion, and the second one specifies the combinatorics of P.

In this paper we study skeletal cut loci, obtaining four main results. First, given any combinatorial tree T without degree-2 nodes, there exists a convex polyhedron P and a point x in P with a cut locus that lies in $\mathrm{Sk}(P)$, and whose combinatorics match T. Second, any (non-degenerate) polyhedron P has at most a finite number of points x for which $\mathcal{C}(x) \subset \mathrm{Sk}(P)$. Third, we show that almost all polyhedra have no skeletal cut locus. Fourth, we provide a combinatorial restriction to the existence of skeletal cut loci.

Because the source unfolding of P with respect to x is always a non-overlapping net for P, and because the boundary of the source unfolding is the (unfolded) cut locus, source unfoldings of polyhedra with skeletal cut loci are edge-unfoldings, and moreover "blooming," avoiding self-intersection during an unfolding process.

1 Introduction

Our focus is the cut locus $\mathcal{C}(x)$ on a convex polyhedron, and the relationship of $\mathcal{C}(x)$ to the 1-*skeleton* of P—the graph of vertices and edges—which we denote by $\mathrm{Sk}(P)$. The *cut locus* $\mathcal{C}(x)$ of $x \in P$ is the closure of the set of points on P to which there is more than one geodesic segment (shortest path) from x. $\mathcal{C}(x)$ is a tree whose leaves are vertices of P. Nodes of degree $k \geq 3$ are *ramification points* to which there are k distinct geodesic segments from x. Nodes v of degree 2 in $\mathcal{C}(x)$ can also occur, if v is a vertex of P. For details, see Section 2.1.

The 1-skeleton of a non-degenerate polyhedron is a 3-connected graph by Steinitz's theorem. We call a doubly-covered convex polygon a *degenerate* convex polyhedron, for which the 1-skeleton is a cycle. We say that P has a *skeletal cut locus* if there is some $x \in P$ such that $\mathcal{C}(x) \subset \mathrm{Sk}(P)$.

The edges of $\mathcal{C}(x)$ are known to be geodesic segments [AAOS97], so it is at least conceivable that an edge of $\mathcal{C}(x)$ lies along an edge of P. Theorem 1 shows that, for certain polyhedra P and points $x \in P$, all of $\mathcal{C}(x)$ lies in the 1-skeleton of P: $\mathcal{C}(x) \subset \mathrm{Sk}(P)$. As a simple example, we will see in Lemma 7 that the three edges incident to any vertex of a tetrahedron form $\mathcal{C}(x)$ for an appropriate x, and are therefore a skeletal cut locus.

Although Theorems 6 and 8 will show that skeletal cut loci are "rare" in senses we'll make precise, Theorem 1 and its proof establish that uncountably many polyhedra do admit skeletal cut loci, in a sense made quantitatively precise by Proposition 4.

Theorem 1 can also be viewed as a companion to the main result in [OV23], that any *length* (or *metric*) *tree*—a tree with specified edge lengths—can be realized as the cut locus on some polyhedron. Here we only match the combinatorics of T, not its metrical properties, but requiring additionally for T to be included in $\mathrm{Sk}(P)$.

Connection to Unfolding. It has long been known that cutting the cut locus $\mathcal{C}(x)$ and unfolding to the plane leads to the non-overlapping *source unfolding*: If x is not itself at a vertex, then the unfolding arrays all the shortest paths 2π around x (because x is surrounded by 2π of surface), with the image of the cut locus forming the boundary of the unfolding [Mou85] [SS86]. If x is a vertex, then the shortest paths from x cover a wedge of the total surface angle at x. For the polyhedra in Theorem 1, the source unfolding is an edge-unfolding. And because it is known that the source unfolding can be *bloomed*—unfolded continuously from $\mathbb{R}^3$ to $\mathbb{R}^2$ without self-intersection [DDH+11]—Theorem 1 and its companion Proposition 4 provide perhaps the first infinite class of examples of blooming edge-unfoldings. It remains unknown whether every non-overlapping edge-unfolding can be bloomed.

A central open problem in our work asks for an accounting of all the polyhedra P that support a skeletal cut locus. All of these enjoy the property that source unfoldings are also blooming edge-unfoldings.

2 Construction of Skeletal Cut Loci

Our first result is the following theorem.

*The full version of this paper is [OV24b].

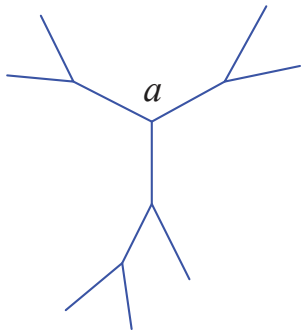


Figure 1: Tree T with 7 leaves.

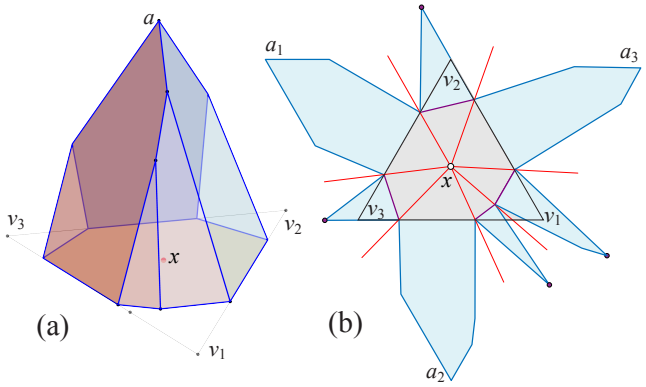


Figure 2: (a) P is created from a regular tetrahedron by four vertex truncations. $\mathcal{C}(x)$ consists of all non-base edges, and is homeomorphic to the tree in Fig. 1. (b) Source unfolding of P from x. Bisectors shown red.

Theorem 1 *Given any combinatorial tree T without degree-2 nodes there is a convex polyhedron P and a point $x \in P$ such that the cut locus $\mathcal{C}(x)$ is entirely contained in* $\mathrm{Sk}(P)$*, and the combinatorics of $\mathcal{C}(x)$ match T.*

We first illustrate the main idea of the construction before addressing details. Suppose the given tree T is the 7-leaf tree shown in Fig. 1. We select a degree-3 node as root a, which corresponds to the apex of a regular tetrahedron $av_1v_2v_3$. We fix x at the centroid of the base Q.

Fig. 2(a) show one possible construction of P. The edges incident to a are clearly in $\mathcal{C}(x)$ with x at the centroid of the base triangle. All three base vertices of the tetrahedron are then truncated, with the truncation of v_1 being followed by a truncation of one of the two base vertices created. Now T corresponds to all the non-base edges of P.

The truncations are not arbitrary: the truncation planes must have precise tilts in order for the edges of each truncation to lie in $\mathcal{C}(x)$. Fig. 2(b) shows the source unfolding of P, with a_1, a_2, a_3 the three images of a. The red bisector rays from x through the truncation vertices on the base Q suggest that indeed any point p on a truncation edge is equidistant from x and therefore on $\mathcal{C}(x)$.

Returning to the need for precise tilts of the tuncation planes, let z be the point on the edge av_1 through which the truncation plane passes, creating a truncation triangle zt_1t_2. As indicated in Fig. 3, the tilt is uniquely determined by the location of z: the placement of z determines t_1, t_2, and the edge t_1t_2 determines z.

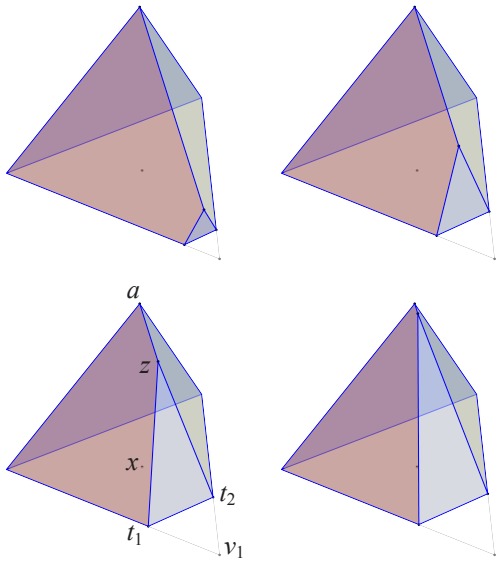


Figure 3: The tilt of the truncation plane is determined by the position of z on av_1.

2.1 Cut Locus Preliminaries

For the readers convenience, we list next several basic properties of cut loci, sometimes used implicitly in the following.

(i) $\mathcal{C}(x)$ is a tree drawn on the surface of P. Its leaves are vertices of P, and all vertices of P, excepting x (if it is a vertex) are included in $\mathcal{C}(x)$. All points interior to $\mathcal{C}(x)$ of degree 3 or more are known as *ramification points* of $\mathcal{C}(x)$. All vertices of P interior to $\mathcal{C}(x)$ are also considered as ramification points, of degree at least 2; see e.g. Fig. 7.

(ii) Each point y in $\mathcal{C}(x)$ is joined to x by as many geodesic segments as the number of connected components of $\mathcal{C}(x) \setminus y$. For ramification points in $\mathcal{C}(x)$, this is precisely their degree in the tree.

(iii) The edges of $\mathcal{C}(x)$ are geodesic segments on P.

(iv) Assume the distinct geodesic segments γ and γ' from x to $y \in C(x)$ bound a domain D of P, which

intersects no other geodesic segment from x to y. Then there is an arc of $\mathcal{C}(x)$ at y which intersects D and bisects the angle of D at y.

(v) The tree $\mathcal{C}(x)$ is reduced to a path, if and only if the polyhedron is a doubly-covered (planar) convex polygon, with x on the rim.

Further details and references can be found in [OV24a, Ch. 2].

2.2 Construction Details

Throughout we assume T has no degree-2 nodes. Start with P a pyramid with apex a centered over a regular n-gon base Q, with x the centroid of Q. Label the vertices of Q as $v_1, \ldots, v_n$.

The construction does not depend on the degree of apex a, so it is no loss of generality to assume a has degree-3 so that P starts as a regular tetrahedron. Let z be a node of T adjacent to a. (We will often use a and z and other variables to both refer to a node of T and a corresponding vertex of P.) Let z have degree $k+2$ in T. Truncation by k planes through z will create a vertex at z of degree $k+2$. E.g., if z is degree-3, $k=1$ plane through z creates a vertex of degree-3, as we've seen in Fig. 3.

We aim to understand how to truncate by $k \geq 1$ planes through z so that the $k+1$ truncation edges from z incident to the base Q are part of $\mathcal{C}(x)$. We will illustrate in detail the case $k=2$ shown in Fig. 4. Looking ahead, if we know how to construct k planes through z, then we can apply the same logic to construct j planes through a child y of z. The $j=1$ case is illustrated in Fig. 5, with the red truncation triangle incident to y. Then the same construction technique can be used to inductively create the full subtree rooted at z. We will show later that the subtrees rooted at the other two children of a can be arranged to avoid interfering with one another.

We express the construction as a multi-step algorithm, and later prove that the truncation edges are in $\mathcal{C}(x)$. Fix $k \geq 1$, and position z anywhere in the interior of av_1. The goal is to compute the *truncation chain* $t_1, t_2, \ldots, t_k, t_{k+1}$ on base Q, where $t_1 \in v_1v_3$ and $t_{k+1} \in v_1v_2$ (e.g., t_1, t_2, t_3 in Fig. 4). Each truncation triangle is then zt_it_{i+1}.

The construction of the truncation chain is effected by first computing the unfolded positions z_i, the images of z in the unfolding. It is perhaps counterintuitive, but we can calculate z_i without knowing t_it_{i+1}; instead we use z_i to calculate t_it_{i+1}. The next construction depends on our choice of several parameters; we'll see later that it provides a suitable polyhedron.

(1) z_0 is the position of z after unfolding the left face of the tetrahedron about v_3v_1 to the base plane. z_0 can be determined by $|v_1z| = |v_1z_0|$. Then z_{k+1} is the reflection of z_0 across xv_1.

(2) Set $r_z = |xz_0| = |xz_{k+1}|$.

(3) All the z_i's are chosen to lie on the circle C_z centered on x of radius r_z.

(4) Let A be the angle z_0xz_{k+1}. Partition A into $k+1$ angles α. This is another choice, to maximize the symmetry of the construction.

(5) The z_i's lie on rays from x separated by α. Together with C_z, this determines the location of the z_i's.

(6) Set B_i to bisect the angle at x between the z_{i-1}, z_i rays, $i = 1, \ldots, k+1$.

(7) We determine t_1 and t_{k+1} using the first and last bisector: $t_1 = v_1v_3 \cap B_1$, $t_{k+1} = v_1v_2 \cap B_{k+1}$. The intermediate chain vertices $t_2, \ldots, t_k$ are not yet determined.

(8) Let Π_i be the mediator plane through zz_i, the plane orthogonal to zz_i through its midpoint. It is these planes that determine t_i, $i = 2, \ldots, k$.

(9) Π_i intersects the xy-plane in a line L_i containing t_it_{i+1}.

(10) $t_i = L_i \cap B_i$.

First note that the mediator plane construction of t_it_{i+1} guarantees that z unfolds to z_i. Second, the angles between edges t_iz_{i-1} and t_iz_i are split by B_i by construction. So any point p on the interior of edge zt_i unfolds to two images in the plane equidistant from x.

Lemma 2 *Each truncation edge zt_i is an edge of $\mathcal{C}(x)$.*

Proof. We first prove that zt_1 lies in $\mathcal{C}(x)$. Throughout refer to Fig. 6.

Before truncation, the segment zt_1 lies on the face av_3v_1 of the polyhedron P, which is a regular tetrahedron in this case.

Fix a point $p \in zt_1$. The unique shortest path γ to p crosses edge v_1v_3. After truncation, γ remains a geodesic arc. We aim to prove that it remains shortest, and moreover there is another companion geodesic segment γ', establishing that $p \in \mathcal{C}(x)$.

Now we consider the situation after truncation. Let δ be a geodesic arc from x to p, approaching p from the other side of zt_1; see Fig. 6(b). If δ crosses the edge t_1t_2, then we have $|\gamma| = |\delta|$ by construction, and we have found $\gamma' = \delta$.

Suppose instead that δ crosses edge t_it_{i+1} for $i \geq 2$, and then crosses the truncation triangles $zt_it_{i+1}, zt_{i-1}t_i, \ldots, zt_1t_2$ (right to left, i.e., clockwise, in Fig. 6(a)) before reaching p. To simplify the discussion,

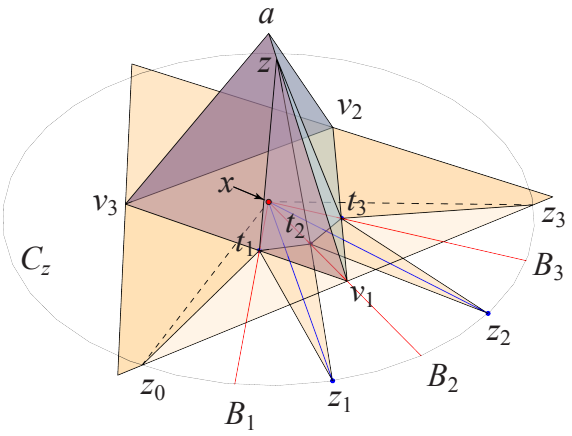


Figure 4: $k = 2$ truncation planes through z.

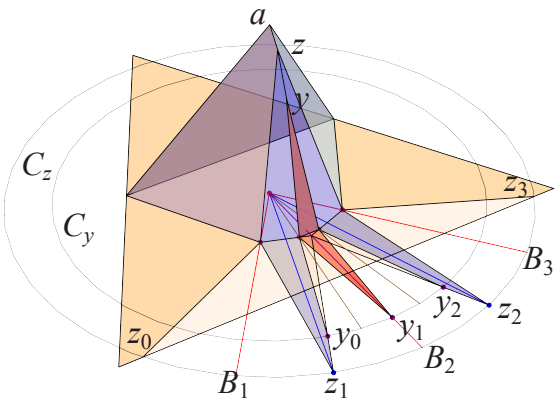


Figure 5: $k = 2$, $j = 1$. The y-truncation cuts the zt_2 edge in Fig. 4.

we illustrate $i = 2$, so δ crosses t_2t_3 and then triangles zt_2t_3 and zt_1t_2. See Fig. 6(b).

Let q_2 be the quasigeodesic[1] xt_2z on P'; it must be crossed by δ to reach p. There are two triangles xt_2z_1 and xt_2z_2 bounding q_2 to either side, congruent by the construction. Thus the construction has local intrinsic symmetry about q_2.

Let s be the point at which δ crosses t_2t_3, $\{s\} = \delta \cap t_2t_3$. First assume that s lies in the triangle xt_2z_2. Then δ remains in xt_2z_2 until it crosses q_2. Then there must be another geodesic arc δ' symmetric with δ about q_2, as illustrated in (b). So δ and δ' meet at a point of q_2. Because δ and δ' have the same length, neither can be a shortest path beyond that point of intersection. Therefore δ cannot reach p as a geodesic segment.

Second, if s instead lies in the triangle xt_3z_2, then it is clear from the planar image in (a) of the figure that δ cannot cross the segment xz_2 clockwise, which it must to reach p from the right in the figures. So δ must head counterclockwise, crossing $q_3 = xt_3z$. Then

[1] A *quasigeodesic* is a path with at most π surface to either side of every point.

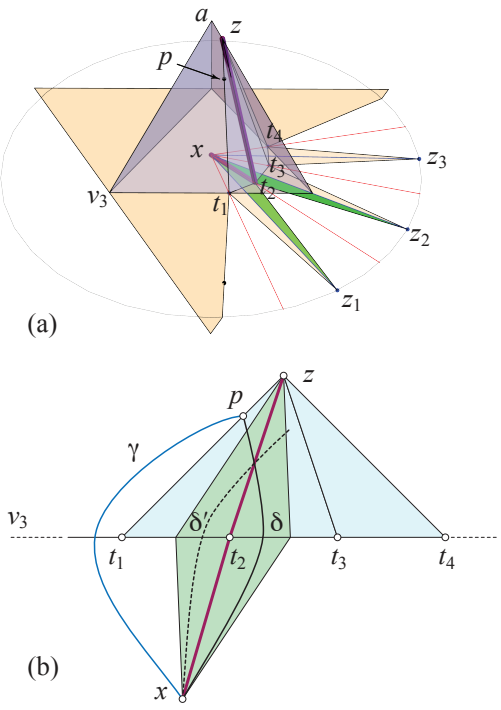


Figure 6: Proof that $p \in zt_1$ is on $\mathcal{C}(x)$. (a) Quasigeodesic $q_2 = xt_2z$ shown purple and congruent triangles xt_2z_1 and xt_2z_2 shaded green. (b) Abstract picture depicting geodesic segments γ, δ, δ'.

the same argument applies, based this time on the local intrinsic symmetry about q_3, and shows that δ cannot be a shortest path beyond q_3.

We have established that every point p on zt_1 is on $\mathcal{C}(x)$, and so $zt_1 \subset \mathcal{C}(x)$. The same argument applies to zt_{k+1}, the rightmost truncation edge in the figures.

So now we know that two geodesic segments from x to z cross t_1t_2 and t_kt_{k+1}. These two segments determine a digon D within which the remaining segments of $\mathcal{C}(x)$ lie. But within D we have local intrinsic symmetry with respect to the quasigeodesics $q_i = xt_iz$, because q_i is surrounded by the congruent triangles xt_iz_{i-1} and xt_iz_i. Therefore, the previous argument shows that all the edges zt_i are included on $\mathcal{C}(x)$. □

We now return to the claim that the three subtrees descendant from a do not interfere with one another.

Lemma 3 *The truncations for one subtree descendant of apex a do not interfere with another subtree descendant.*

Proof. First, as $k \to \infty$, t_1 approaches the line xz_0. Thus the leftmost truncation triangle stays to the v_1-side of the midpoint of v_1v_3, say by ε. Second, subsequent truncations to all but the extreme edges zt_1 and zt_{k+1} stay inside the $t_1, \ldots, t_k$ chain. The only concern would be that truncation of the zt_1 edge crossed the

midpoint of v_1v_3 (and so possibly interfering with truncations of av_3). However, as is evident in the earlier Fig. 3, the position of t_1 moves monotonically toward v_1 as z moves down av_1. Thus we can widen ε to accommodate a truncation of zt_1 (or of zt_{k+1}). So the entire subtree rooted at z stays between the midpoints of v_1v_3 and v_1v_2. □

Further examples are shown in [OV24b].

Lemmas 2 and 3 together establish Theorem 1: $\mathcal{C}(x) \subset \mathrm{Sk}(P)$ matches the given T.

3 Theorem 1 Discussion

We mentioned in Section 1 that Theorem 1 leads to an uncountable number of skeletal polyhedra. This follows immediately from the freedom to place z at any point interior to av_1 in the construction detailed in Section 2.2. We can be more quantitatively precise, as follows.

Assume that T is a cubic tree without degree-2 nodes, so it has n leaves and $n-2$ ramification points. Aside from one ramification point, which is chosen as the apex of the starting tetrahedron, all others are free to vary on their respective edges in our construction, which implies $n-3$ free parameters. Because $\mathcal{C}(x)$ is skeletal, each ramification point of T is a vertex of P, so P has $V = 2n-2$ vertices, and $n = V/2+1$. The space $\mathfrak{P}_V$ of all convex polyhedra with V vertices, up to isometries, has dimension $3V-6$ (see for example [LP22]), hence the starting tetrahedron provides another 6 free parameters and we have the next result.

Proposition 4 *The set of convex polyhedra admitting skeletal cut loci—and hence blooming edge-unfoldings—contains a subset of dimension $\geq V/2+4$ in the $(3V-6)$-dimensional space of all convex polyhedra with V vertices, up to isometries.*

Recall we restricted Theorem 1 to trees T without degree-2 nodes. Our construction can be viewed as realizing degree-2 nodes of T with flat "vertices" on $\mathrm{Sk}(P)$—points interior to edges of P. We are currently extending the construction to match degree-2 nodes of T with non-flat vertices of P.

Our construction for Theorem 1 results in a *dome*, a convex polyhedron P with a distinguished base face Q, with every other face sharing an edge with Q. It was already known that domes have edge-unfoldings [DO07, p. 325], although the proof of non-overlapping for our domes is almost trivial—the source unfolding does not overlap.

Although our previous construction results in domes, there are many other polyhedra with skeletal cut loci, see e.g. Fig. 7 and Theorem 9. Which leaves us with this central open problem: *Characterize all convex polyhedra P which admit skeletal cut loci.* The remainder of the paper addresses and partially answers this problem.

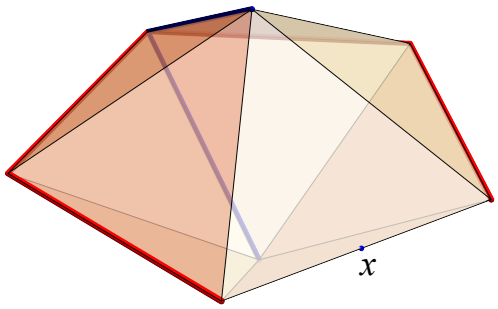


Figure 7: P: pentagonal dipyramid. $\mathcal{C}(x)$: red and blue edges of $\mathrm{Sk}(P)$.

Several natural questions now suggest themselves:

(1) For a fixed P, how many distinct points x can lead to skeletal cut loci? (Theorem 6).

(2) Can all of $\mathrm{Sk}(P)$ for a given P be covered by skeletal cut loci? (Proposition 5).

(3) How common / rare are skeletal cut loci in the space of all convex polyhedra? (Theorem 8).

(4) Are there restrictions for the existence of skeletal cut loci? (Proposition 5, Theorems 6 and 10).

Due to space limitations, we cite next lemmas and theorems often without motivations or proofs, all of which may be found in [OV24b].

4 Existence of Several Skeletal Cut Loci

In the first two questions in the list above, degenerate P play a special role:

Proposition 5

(a) There exists infinitely many points x with $\mathcal{C}(x) \subset \mathrm{Sk}(P)$ if and only if P is degenerate.

(b) There exists two points x_1, x_2 on P whose cut loci together cover $\mathrm{Sk}(P)$ if and only if P is degenerate.

Example 1 *Consider a regular dipyramid P over a convex $2m+1$-gon; see Fig. 7. One can see that, for every midpoint x of a "base edge" e, $\mathcal{C}(x)$ is included in $\mathrm{Sk}(P)$. More precisely, $\mathcal{C}(x)$ contains all base edges other than e, and the two "lateral edges" opposite to x. In particular, this provides $2m+1$ such points, for $V = 2m+3$ vertices.*

Theorem 6 *For any non-degenerate convex polyhedron P with E edges, there are at most $2\binom{E}{2}$ flat points x of P such that $\mathcal{C}(x) \subset \mathrm{Sk}(P)$.*

5 Absence of Skeletal Cut Loci

Lemma 7 *Every tetrahedron T has four points $x \in T$ such that $\mathcal{C}(x) \subset \mathrm{Sk}(T)$.*

Proof. For each vertex v_i, denote by x_i the ramification point of $\mathcal{C}(v_i)$. It follows, from cut locus Property (ii), that v_i is the ramification point of $\mathcal{C}(x_i)$. Then, by (i) and (iii), $\mathcal{C}(x_i)$ consists of the three edges incident to v_i. □

Theorem 8 *For almost all*[2] *convex polyhedra P with $V > 4$ vertices, there exists no point $x \in P$ with $\mathcal{C}(x) \subset \mathrm{Sk}(P)$.*

Proof. Notice first that almost all convex polyhedra P are non-degenerate.

Assume, for the simplicity of the exposition, that every face of P is a triangle and $\mathrm{Sk}(P)$ is a cubic graph.

Case 1. Assume there exists a flat point x interior to some face F of P, such that $\mathcal{C}(x) \subset \mathrm{Sk}(P)$.

Repeating the notation in Theorem 6, denote by v_i, $i = 1, 2, 3$, the vertices of F, and by e_i the edges of P incident to v_i and not included in F. Moreover, denote by γ_i the geodesic segment from x to v_i.

As in Theorem 6, it follows that $e_i \subset C(x)$ so, together, γ_i and e_i bisect the complete angle at v_i. In other words, the straight extensions E_i into F by all the e_i are concurrent: they all intersect at the same point.

Now we perturb the vertices of P to destroy this concurrence. If P were a tetrahedron, then perturbing the apex would simultaneously move the edges incident to it. But the assumption that $V > 4$ means that there are at least two vertices outside the 3-vertex face F containing x. Perturbing these two vertices independently moves the edges incident to F independently, breaking the concurrence at x.

Because there are at most finitely many such points x by Theorem 6, the conclusion follows in this case.

Case 2. Assume there exists a flat point x interior to some edge e of P, such that $\mathcal{C}(x) \subset \mathrm{Sk}(P)$. Denote by v_i, $i = 1, 2$, the vertices of e, and by e_i the edges of P incident to v_i included in $\mathcal{C}(x)$. As above, it follows that the straight extensions of e_1, e_2 coincide with e. Now, small perturbations of the vertices of P destroy this coincidence. Note that if e, e_1, e_2 form a triangle, then e_1, e_2 will move together. But still, perturbations at other vertices of P (not $v_1, v_2, e_1 \cap e_2$) will destroy the concurrence.

[2] I.e., polyhedra in an open and dense subset of $\mathfrak{P}_V$.

Case 3. Assume finally there exists a vertex v of P, such that $\mathcal{C}(v) \subset \mathrm{Sk}(P)$. Here we obtain again that the straight extensions of two edges contain (other) edge-pair extensions, and small perturbations of the vertices of P destroy this coincidence. □

6 Every Vertex a Skeletal Source

Theorem 9 *Assume that every vertex of P has a skeletal cut locus. Then the following statements hold.*

1. *Every face of P is a triangle.*
2. *Every vertex of P has even degree in $\mathrm{Sk}(P)$.*
3. *The edges at every vertex v split the complete angle at v into evenly many sub-angles, every two opposite such angles being congruent.*
4. *If, moreover, every vertex of P has degree 4 in $\mathrm{Sk}(P)$ then P is an octahedron:*
   - *with three planar symmetries, and*
   - *all faces of which are acute congruent (but not necessarily equilateral) triangles.*

Example 2 *Suitable dipyramids over convex $2m$-gons, similar to Example 1, provide non-octahedron polyhedra whose the cut loci of the vertices cover the 1-skeleton.*

7 A Combinatorial Restriction

Already mentioned in the Abstract, at a first glance there seems to be very little relation between the cut locus and the 1-skeleton, as the first one is an intrinsic geometry notion, and the second one specifies the combinatorics of P. A background connection between the two notions can however be established in two steps: Alexandrov's Gluing Theorem connects the intrinsic and the extrinsic geometry of P, while Steinitz's Theorem relates the combinatorics to the extrinsic geometry.

In this section we provide an easy combinatorial restriction to the existence of skeletal cut loci, complementing the first part of Theorem 9.

Lemma 2.8 in [OV24a] shows that, at a vertex v of P of degree-3 in $\mathrm{Sk}(P)$, the sum of any two face angles incident to v is strictly larger than the third angle. We now argue that such a v cannot be a degree-2 node in a cut locus. Assume otherwise. Then there are precisely two geodesic segments from x to v, and they form two angles around v. By (iv), each of the two edge-branches of $C(x)$ starting at v will bisect one of those angles. Then the angles at v to the left and to the right of $C(x)$ are equal, impossible by the mentioned Lemma 2.8.

In the literature, a spanning tree without degree-2 nodes is called a *HIST*.[3] So every spanning tree of a HIST-free graph has a degree-2 node. Because a degree-3 vertex cannot be a degree-2 node in a cut locus, we have the following combinatorial restriction.

Theorem 10 *A HIST-free cubic polyhedral graph cannot be realized with skeletal cut loci.*

One can check straightforwardly that, among the Platonic solids, the cube and the dodecahedron graphs are HIST-free, hence these polyhedra do not admit skeletal cut loci.

Acknowledgements. We benefited from the suggestions of three reviewers, and we thank Joseph Malkevitch for information on HISTs.

[3]HIST abbreviates "homeomorphically irreducible spanning tree." See, e.g., [GNRZ24] and the references therein.

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.

[DDH+11] Erik D. Demaine, Martin L. Demaine, Vi Hart, John Iacono, Stefan Langerman, and Joseph O'Rourke. Continuous blooming of convex polyhedra. *Graphs and Combinatorics*, 27:363–376, 2011.

[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007. http://www.gfalop.org.

[GNRZ24] Jan Goedgebeur, Kenta Noguchi, Jarne Renders, and Carol T. Zamfirescu. HIST-critical graphs and Malkevitch's conjecture. arXiv:2401.04554., 2024.

[LP22] Nina Lebedeva and Anton Petrunin. Alexandrov's embedding theorem. arXiv:2212.10479, 2022.

[Mou85] David M. Mount. On finding shortest paths on convex polyhedra. Technical Report 1495, Dept. Computer Science, Univ. Maryland, 1985.

[OV23] Joseph O'Rourke and Costin Vîlcu. Cut locus realizations on convex polyhedra. *Comput. Geom.: Theory & Appl.*, 114, 2023. doi.org/10.1016/j.comgeo.2023.102010.

[OV24a] Joseph O'Rourke and Costin Vîlcu. *Reshaping Convex Polyhedra*. Springer, 2024. doi.org/10.1007/978-3-031-47511-5.

[OV24b] Joseph O'Rourke and Costin Vîlcu. Skeletal cut loci on convex polyhedra. arXiv:2312.01534v2 [cs.CG], February 2024.

[SS86] Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. *SIAM J. Computing*, 15(1):193–215, 1986.

Dispersive Vertex Guarding for Simple and Non-Simple Polygons*

Sándor P. Fekete† Joseph S. B. Mitchell‡ Christian Rieck§ Christian Scheffer¶ Christiane Schmidt‖

Abstract

We study the Dispersive Art Gallery Problem with vertex guards: Given a polygon $\mathcal{P}$, with pairwise geodesic Euclidean vertex distance of at least 1, and a rational number ℓ; decide whether there is a set of vertex guards such that $\mathcal{P}$ is guarded, and the minimum geodesic Euclidean distance between any two guards (the so-called *dispersion distance*) is at least ℓ.

We show that it is NP-complete to decide whether a polygon with holes has a set of vertex guards with dispersion distance 2. On the other hand, we provide an algorithm that places vertex guards in *simple* polygons at dispersion distance at least 2. This result is tight, as there are simple polygons in which any vertex guard set has a dispersion distance of at most 2.

1 Introduction

The Art Gallery Problem is one of the fundamental challenges in computational geometry. It was first introduced by Klee in 1973 and can be stated as follows: Given a polygon $\mathcal{P}$ with n vertices and an integer k; decide whether there is a set of at most k many guards, such that these guards see all of $\mathcal{P}$, where a guard sees a point if the line segment connecting them is fully contained in the polygon.

Chvátal [4] and Fisk [8] established tight worst-case bounds by showing that $\lfloor n/3 \rfloor$ many guards are sometimes necessary and always sufficient. On the algorithmic side, Lee and Lin [11] proved NP-hardness; more recently, Abrahamsen, Adamaszek, and Miltzow [1] showed $\exists\mathbb{R}$-completeness, even for simple polygons.

*Work at TU Braunschweig is partially supported by the German Research Foundation (DFG), project "Computational Geometry: Solving Hard Optimization Problems" (CG:SHOP), grant FE407/21-1. J. S. B. Mitchell is partially supported by the National Science Foundation (CCF-2007275). C. Schmidt is partially supported by Vetenskapsrådet by the grant "ILLUMINATE provably good methods for guarding problems" (2021-03810).

†Department of Computer Science, TU Braunschweig, s.fekete@tu-bs.de

‡Department of Applied Mathematics and Statistics, Stony Brook University, joseph.mitchell@stonybrook.edu

§Department of Computer Science, TU Braunschweig, rieck@ibr.cs.tu-bs.de

¶Faculty of Electrical Engineering and Computer Science, Bochum University of Applied Sciences, christian.scheffer@hs-bochum.de

‖Department of Science and Technology, Linköping University christiane.schmidt@liu.se

In this paper, we investigate the Dispersive AGP in polygons with vertex guards: Given a polygon $\mathcal{P}$ and a rational number ℓ, find a set of vertex guards such that $\mathcal{P}$ is guarded and the minimum pairwise geodesic Euclidean distance between each pair of guards is at least ℓ. (Note that the cardinality of the guard set does not come into play.)

1.1 Our Contributions

We give the following results for the Dispersive Art Gallery Problem in polygons with vertex guards.

- For polygons with holes, we show NP-completeness of deciding whether a pairwise geodesic Euclidean distance between any two guards of at least 2 can be guaranteed.
- For simple polygons, we provide an algorithm for computing a set of vertex guards of minimum pairwise geodesic distance of at least 2.
- We show that a dispersion distance of 2 is worst-case optimal for simple polygons.

1.2 Previous Work

Many variations of the classic Art Gallery Problem have been investigated [13, 15, 16]. This includes variants in which the number of guards does not play a role, such as the Chromatic AGP [6, 7, 10] as well as the Conflict-Free Chromatic AGP [2, 3, 9].

The Dispersive AGP was first introduced by Mitchell [12], and studied for the special case of polyominoes by Rieck and Scheffer [14]. They gave a method for computing worst-case optimal solutions with dispersion distance at least 3 for simple polyominoes, and showed NP-completeness of deciding whether a polyomino with holes allows a set of vertex guards with dispersion distance of 5.

1.3 Preliminaries

Given a polygon $\mathcal{P}$ (possibly with holes), we say that two points $p, q \in \mathcal{P}$ *see each other*, if the connecting line segment $\overline{pq}$ is fully contained in $\mathcal{P}$. A (finite) set of points $\mathcal{G} \subset \mathcal{P}$ is called a *guard set* for $\mathcal{P}$, if all points of $\mathcal{P}$ are seen by at least one point of $\mathcal{G}$. If $\mathcal{G}$ is a subset of the vertices of $\mathcal{P}$, we are dealing with *vertex guards*.

Distances between two points $p, q \in \mathcal{P}$ are measured according to the Euclidean geodesic metric, i.e., that is the length of a shortest path between p and q that stays fully inside of $\mathcal{P}$, and are denoted by $\delta(p, q)$. The smallest distance between any two guards within a guard set is called its *dispersion distance*.

2 First Observations

We start with two easy observations; the second resolves an open problem by Rieck and Scheffer [14], who raised the question about the ratio of the cardinalities of guard sets in optimal solutions for the Dispersive AGP and the classical AGP.

2.1 Shortest Polygon Edges Are Insufficient as Lower Bounds

To see that an optimal dispersion distance may be considerably shorter than the shortest polygon edge, consider Figure 1. Every edge in the polygon has similar length (say, between 1 and $1+\varepsilon$). To guard the colored regions, one of each of the same colored vertices needs to be in the guard set. This results in two guards that are arbitrarily close to each other.

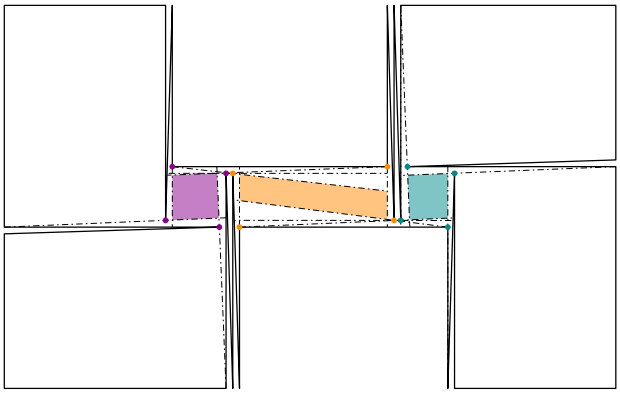


Figure 1: A polygon in which edges have similar length.

This motivates our assumption that the geodesic distance between any pair of vertices is at least 1.

2.2 Optimal Solutions May Contain Many Guards

Even for a polygon that can be covered by a small number of guards, an optimal solution for the Dispersive AGP may contain arbitrarily many guards; see Figure 2. An optimal solution for the classical AGP consists of 2 guards placed at both ends of the central edge of length ε. On the other hand, we can maximize the dispersion distance in a vertex guard set by placing one guard at the tip of each of the $(n-2)/2$ spikes. These two sets have a dispersion distance of ε and 2ζ, respectively, and the ratio $2\zeta/\varepsilon$ can be arbitrarily large.

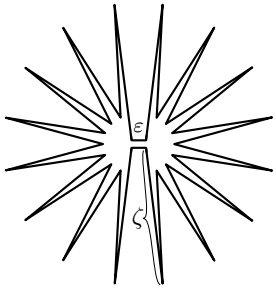


Figure 2: A polygon for which the optimal guard numbers for AGP and Dispersive AGP differ considerably.

3 NP-Completeness for Polygons with Holes

We now study the computational complexity of the Dispersive AGP for vertex guards in non-simple polygons.

Theorem 1 *It is* NP*-complete to decide whether a polygon with holes and geodesic vertex distance of at least* 1 *allows a set of vertex guards with dispersion distance* 2*.*

We first observe that the problem is in NP. For a potential guard set $\mathcal{G}$, we can check the geodesic distance between any pair of vertices $g_1, g_2 \in \mathcal{G}$ as follows. Because any two polygon vertices have mutual distance of at least 1, a shortest geodesic path between g_1 and g_2 consisting of at least two edges has a length of at least 2. This leaves checking the length of geodesic paths consisting of a single edge, which is straightforward.

3.1 Overview and Gadgets

For showing NP-hardness, we utilize the NP-complete problem Planar Monotone 3Sat [5], which asks for the satisfiability of a Boolean 3-CNF formula, for which the literals in each clause are either all negated or all unnegated, and the corresponding variable-clause incidence graph is planar.

To this end, we construct gadgets to represent (i) variables, (ii) clauses, (iii) a gadget that splits the respective assignment, and (iv) gadgets that connect subpolygons while maintaining the given truth assignment.

Variable Gadget. A *variable gadget* is shown in Figure 3. Its four vertices v_1, v_2, v_3, v_4 are placed on the vertices of a rhombus (shown in grey) formed by two adjacent equilateral triangles of side length 1. We add two sharp spikes by connecting two additional vertices (v_5 and v_6) to the two pairs v_1, v_3 and v_2, v_4, respectively; the edges $\{v_3, v_5\}, \{v_2, v_6\}$ have unit-length. (The function of these spikes is to impose an upper bound of 2 on the achievable distance.) We also attach two narrow polygonal corridors to two other pairs of vertices, indicated in green for the pair v_1, v_2, and in red for v_3, v_4. These corridors have appropriate width, up to 1, at the other end, to attach them to other gadgets.

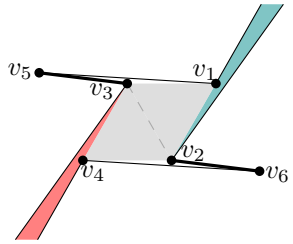


Figure 3: A variable gadget.

Lemma 2 *Exactly four vertex guard sets realize a dispersion distance of* 2 *to guard the subpolygon* $\mathcal{P}_v = (v_1, v_2, v_6, v_4, v_3, v_5, v_1)$ *of a variable gadget.*

Proof. As shown in Figure 3, at least one guard has to be placed on one of $\{v_1, v_2, v_3, v_4\}$ to guard the rhombus that represents the variable. Conversely, it is easy to see that the dispersion distance is less than 2 if more than one guard is chosen from $\{v_1, v_2, v_3, v_4\}$. Furthermore, if we choose v_1 or v_3, the spike at v_5 is guarded, and we can choose v_6 (which has distance 2 from both v_1 and v_3) to guard the other spike; conversely, a guard at v_2 or v_4 covers the spike at v_6 and allows a guard at v_5.

Now a guard from $\{v_1, v_2\}$ also covers the green portion of the polygon; this will correspond to setting the variable to `true`. On the other hand, a guard from $\{v_3, v_4\}$ also covers the red portion of the polygon, corresponding to setting the variable to `false`. □

Clause Gadget. A *clause gadget* is depicted in Figure 4. Its three vertices lie on the vertices of an equilateral triangle of side length 1; attached are narrow polygonal corridors, which are nearly parallel to the triangle edges, each using two of the triangle vertices as end points. These corridors have appropriate width, up to 1, at the other end, to attach them to other gadgets.

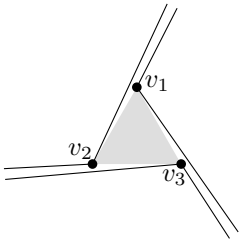


Figure 4: A clause gadget.

Observation 1 *As the vertices* $\{v_1, v_2, v_3\}$ *have a pairwise distance of* 1*, only a single guard can be placed within a clause gadget, if the guard set have to realize a dispersion distance of at least* 2*. A direct consequence is that no more than two of the incident corridors can be guarded by a guard placed on these vertices; hence, at least one corridor needs to be seen from somewhere else, which in turn corresponds to satisfying the clause.*

Split Gadget. A *split gadget* is illustrated in Figure 5. It has one incoming horizontal polygonal corridor, ending at two vertices (v_1 and v_2) within vertical distance 1. These vertices form an equilateral triangle with a third vertex, v_4, where the polygon splits into two further corridors, emanating horizontally from vertices v_3, v_6, and v_5, v_7, respectively. For the upper corridor, the vertices v_1, v_3, v_4, v_6 form slightly distorted adjacent equilateral unit triangles: We move v_3 and v_6 slightly upwards, such that the edges $\{v_1, v_3\}$ and $\{v_4, v_6\}$ as well as the distances between v_1 and v_4 and between v_3 and v_6 remain 1, but the distance between v_3 and v_4 increases to $1 + \varepsilon$. An analogous construction yields the lower outgoing horizontal corridor. Both of these corridors start with a height smaller than 1, but can end with a height of 1 or a very small height.

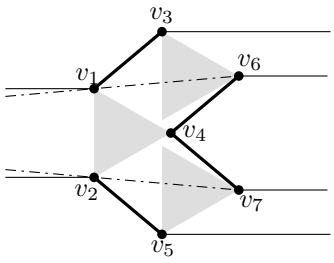


Figure 5: A split gadget.

Lemma 3 *The split gadget correctly forwards the respective variable assignment.*

Proof. We refer to Figure 5 and distinguish two cases. First, assume that the variable adjacent to the left is set to `true`, implying that the connecting corridor is already guarded. Therefore, two guards placed on v_6 and v_7 guard the whole subpolygon, and in particular both corridors to the right.

Now assume that the variable is set to `false`, implying that the corridor to the left is not fully guarded yet. Because this corridor is constructed long enough to contain the intersection of the (dotted) lines through v_1, v_6 and v_2, v_7, we need to place a guard at one of the vertices in $\{v_1, v_2, v_4\}$. Then no further guard can be placed in a distance of at least 2, and the corridors to the right are not guarded, as claimed. □

Connector Gadget. The *connector gadget* is depicted in Figure 6. The distance between all pairs of vertices is at least 1 and less than 2. Furthermore, a guard on either v_1 or v_2 cannot see the horizontal corridor, while a guard on v_3 or v_4 does not see the vertical one.

Observation 2 *The gadget is designed such that only a single guard can be placed on its vertices while maintaining a distance of at least* 2*. If a previously placed*

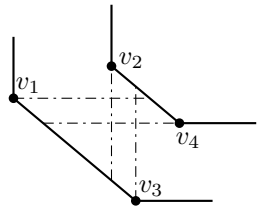


Figure 6: A connector gadget.

guard already sees the vertical corridor, we can place another one to see the horizontal corridor as well. On the other hand, no guard sees both corridors simultaneously. Thus, we propagate a truth assignment.

3.2 Construction and Proof

We now describe the construction of the polygon for the reduction, and complete the proof.

Theorem 1 *It is* NP*-complete to decide whether a polygon with holes and geodesic vertex distance of at least* 1 *allows a set of vertex guards with dispersion distance* 2*.*

Proof. To show NP-hardness, we reduce from Planar Monotone 3Sat. For any given Boolean formula φ, we construct a polygon $\mathcal{P}_\varphi$ as an instance of Dispersive AGP as follows. Consider a planar embedding of the variable-clause incidence graph of φ, place the variable gadgets in a row, and clause gadgets that only consist of unnegated literals or entirely of negated literals to the top or to the bottom of that row, respectively, as illustrated in Figure 7. Furthermore, connect variables to clauses via a couple of connector gadgets, and introduce split gadgets where necessary.

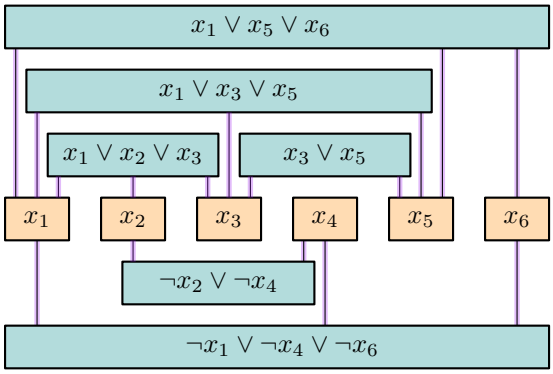


Figure 7: Rectilinear embedding of a Planar Monotone 3Sat instance.

Claim 1 *If φ is satisfiable, then $\mathcal{P}_\varphi$ has a vertex guard set with dispersion distance* 2*.*

Proof. Given a satisfying assignment, we construct a set of vertex guards with a dispersion distance of 2: For every variable that is set to `true`, we place guards on $\{v_1, v_6\}$, and for every variable that is set to `false`, we place guards on $\{v_4, v_5\}$ within the respective variable gadget. Furthermore, we place guards for split and connector gadgets to maintain the given assignments. As we have a satisfying assignment, each clause is satisfied by at least one literal, i.e., at least one corridor incident to the clause gadget is already guarded. Therefore, we can place one guard in each clause gadget. This yields a guard set with a dispersion distance of 2. ■

Claim 2 *If $\mathcal{P}_\varphi$ has a vertex guard set with dispersion distance* 2*, then φ is satisfiable.*

Proof. As we have a set of vertex guards with a dispersion distance of 2, there is only a single guard placed within each clause gadget. Furthermore, no guard set can have larger dispersion distance within a variable gadget. As argued before, there is no guard set with a dispersion distance larger than 2 in the split and connector gadgets. Therefore, the vertex guards placed within the variable gadgets provide a suitable variable assignment for φ. ■

Given that the problem is in NP, these two claims complete the proof. □

4 Worst-Case Optimality for Simple Polygons

In this section we prove that a guard set realizing a dispersion distance of 2 is worst-case optimal for simple polygons. In particular, we describe an algorithm that constructs such guard sets for any simple polygon.

First, we observe that there are polygons for which there is no guard set with a larger dispersion distance.

Observation 3 *There are simple polygons with geodesic vertex distance at least* 1 *for which every guard set has a dispersion distance of at most* 2*.*

Refer to Figure 8. Bold edges have length 1. One of the three vertices (with pairwise distance 1) incident to the gray triangle Δ must be picked to guard Δ, so no guard set can have a dispersion distance larger that 2.

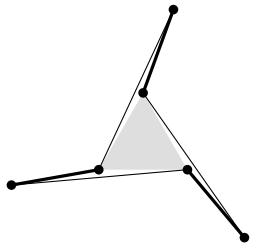


Figure 8: Godfried's favorite polygon.

From this, we can easily obtain polygons with *any* number of vertices of dispersion distance at most 2: Simply modify the polygon at the end of each spike.

In the remainder of this section, we provide a polynomial-time algorithm that constructs a guard set with dispersion distance of at least 2.

We start with a useful lemma that provides some structural properties for the analysis. Refer to Figure 9 for visual reference.

Lemma 4 *Let $\mathcal{P} = (v_1, v_2, \ldots, v_7)$ be a simple polygon with seven vertices labeled in counterclockwise order. Assume that the pairwise (geodesic) distance between all pairs of vertices is at least 1 and further that the following properties are satisfied:*

1. *The distance between v_1 and v_5 is $\delta(v_1, v_5) < 2$.*
2. *v_2, v_4, and v_7 are reflex, i.e., the interior angle at these vertices is strictly larger than $180°$.*

Then the geodesic distance between the two vertices v_3 and v_6 is $\delta(v_3, v_6) \geq 2$.

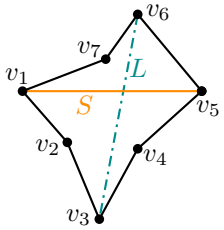


Figure 9: Schematic layout of $\mathcal{P}$.

Proof. Throughout the proof, we will frequently make use of the assumption that $\delta(v_i, v_j) \geq 1$ for any $i \neq j$.

As a first step, we argue that v_1 and v_5 must be mutually visible (along line segment S), as shown in Figure 9: Otherwise, a shortest geodesic path from v_1 to v_5 must visit one of the reflex vertices v_2, v_4, or v_7, implying the contradiction $\delta(v_1, v_5) \geq 2$.

By a similar argument, we claim that v_3 and v_6 are mutually visible (say, along segment L); otherwise we can conclude that $\delta(v_3, v_6) \geq 2$, and we are done.

In the following, we prove that L has length at least 2, by establishing the following two auxiliary claims.

(a) The geodesic distance from v_6 to S is at least $\sqrt{3}/2$.

(b) The geodesic distance from v_3 to S is at least $2 - \sqrt{3}/2 = 1.13397\ldots$.

To this end, assume that the cord S lies horizontally, with $v_1 = (0, 0)$ and $v_5 = (x_5, 0)$, and partitions $\mathcal{P}$ into two subpolygons: (a) the quadrangle $\mathcal{P}' := (v_1, v_5, v_6, v_7)$ above S, and (b) the pentagon $\mathcal{P}'' := (v_1, v_2, v_3, v_4, v_5)$ below S. Because v_7 is reflex, it must lie inside the convex hull of $\mathcal{P}'$, which is spanned by the three remaining vertices v_1, v_5, v_6. Analogously, v_2 and v_4 must lie inside the convex hull of $\mathcal{P}''$, which is spanned by the three remaining vertices v_1, v_3, v_5.

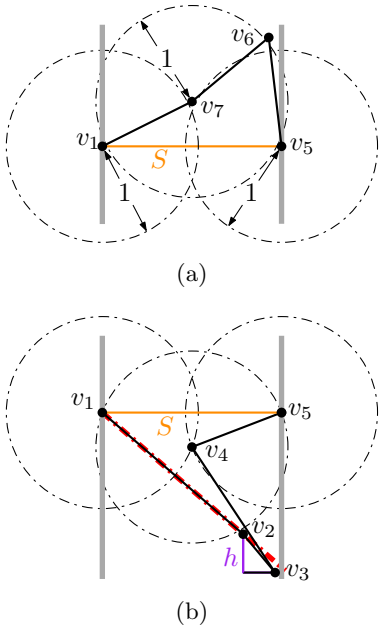


Figure 10: Subpolygons for the proof of the two auxiliary claims: (a) The quadrangle $\mathcal{P}' = (v_1, v_5, v_6, v_7)$. (b) The pentagon $\mathcal{P}'' = (v_1, v_2, v_3, v_4, v_5)$.

For the auxiliary claim (a), refer to Figure 10(a). If $v_6 = (x_6, y_6)$ lies outside the vertical strip defined by $0 \leq x \leq x_5$, then its closest point on S is v_5 (for $x_6 > x_5$) or a point q for which the geodesic to S runs via v_7 (for $x_6 < 0$), so the minimum distance of v_6 is $\delta(v_1, v_6) \geq 1$, (or $\delta(v_5, v_6) \geq 1$, respectively). Therefore, the convex hull of $\mathcal{P}'$ must lie within the strip, including v_7. Furthermore, v_6 must have the largest vertical distance from S, so v_7 must lie within the axis-aligned rectangle R' of height $\sqrt{3}/2$ above S. Consider the three circles C_1, C_5, and C_7 of unit radius around v_1, v_5, and v_7. It is straightforward to verify that R' is completely covered by C_1, C_5, and C_7, implying that v_6 cannot lie inside R', and the first claim follows.

For the auxiliary claim (b), refer to Figure 10(b). Without loss of generality, assume that the vertical distance $-y_2$ of v_2 from S is not smaller than the vertical distance $-y_4$ of v_4. Consider the horizontal positions x_2, x_3, x_4 of v_2, v_3, v_4. Because v_2 lies inside the convex hull of $\mathcal{P}''$, the assumption $x_2 < 0$ (which differs from the figure) implies that $x_3 \leq x_2 < 0$; then the shortest distance from v_2 to S is $\delta(v_2, v_1) \geq 1$, and (because v_2 is reflex), a shortest geodesic path from v_3 to S passes through v_2, so $\delta(v_3, v_1) = \delta(v_3, v_2) + \delta(v_2, v_1) \geq 2$. Analogously, we can assume that $x_4 \leq x_5$. Furthermore, the assumption on the relative vertical positions of v_2 and v_4 implies that v_4 must also lie to the right of v_1, i.e., $x_4 \geq 0$.

Consider $x_2 \geq s$ and refer to Figure 11.

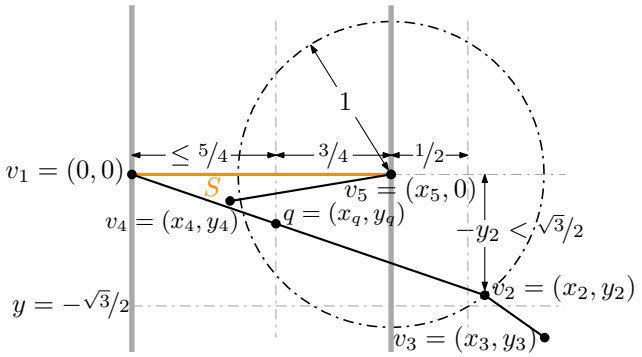


Figure 11: Estimating $\delta(v_2, v_4)$.

Then the assumption $-y_2 \leq \sqrt{3}/2$ (together with $\delta(v_2, v_5) \geq 1$) implies that $x_2 \geq x_5 + 1/2$. Furthermore, v_4 must lie above the edge (v_1, v_2).

We now consider the point $q = (x_5 - 3/4, y_q)$. Because $x_5 < 2$, we conclude $-y_q \leq -y_2/2 \leq \sqrt{3}/4$. Therefore, $\delta(q, v_5) \leq \sqrt{(\frac{4}{3})^2 + (\frac{\sqrt{3}}{4})^2} = 0.96824\ldots < 1$. Because v_4 must lie outside of the circle with radius 1 around v_5, we conclude that $x_4 < x_5 - 3/4$, implying that $\delta(v_2, v_4) \geq 5/4 = 1.25$. Furthermore, v_2 cannot lie on the convex hull of $\mathcal{P}''$, thus, $x_3 > x_2$ and $y_3 < y_2$, implying $\delta(v_3, v_4) > \delta(v_2, v_4)$. As the geodesically shortest path from v_3 to S passes through v_4, we conclude that the length of this path, $\delta(v_3, v_4) - y_4 \geq \delta(v_3, v_4) \geq \delta(v_2, v_4)$ is bounded from below by 1.25. Thus, we can assume that $-x_2 \geq \sqrt{3}/2$ in this case.

Alternatively, consider $x_2 \leq x_5$. Then an argument for v_1, v_2, v_4, v_5 analogous to the one from claim (a) for v_1, v_5, v_6, v_7 also implies that $-x_2 \geq \sqrt{3}/2$.

Consider the vertical distance $h := -y_3 + y_2$ between v_3 and v_2, and refer to Figure 12.

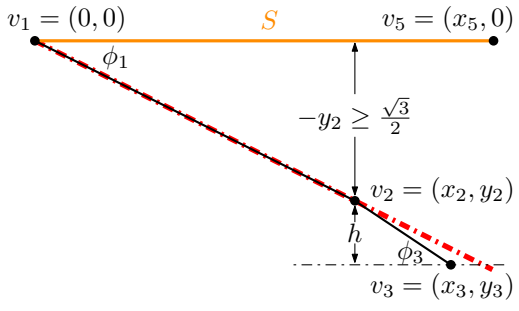


Figure 12: Angles and vertical distances at v_1 and v_3.

To this end, note that the angle ϕ_1 at v_1 between (v_1, v_2) and S satisfies $\tan\phi_1 = -y_2/x_2 \geq \sqrt{3}/4 = \eta$ because of $-y_2 \geq \sqrt{3}/2$ and $x_2 \leq x_5 < 2$. Because v_2 is reflex, the angle ϕ_3 between (v_3, v_2) and a horizontal line at v_3 satisfies $\phi_3 > \phi_1$; moreover, $\sin\phi_3 = \frac{h}{\delta(v_2, v_3)}$, with $\delta(v_2, v_3) \geq 1$, so $h \geq \sin\arctan\eta = \frac{\eta}{\sqrt{1+\eta^2}} = 0.3973\ldots$.

This implies that the vertical distance $-y_3$ of v_3 to S (and thus the distance of v_3 to S) is at least $\sqrt{3}/2 + 0.3973 = 1.26338\ldots > 1.13397\ldots = 2 - \sqrt{3}/2$, as claimed. □

We now show the main result of this section.

Theorem 5 *For every simple polygon $\mathcal{P}$ with pairwise geodesic distance between vertices at least 1, there exists a guard set that has dispersion distance at least 2.*

Proof. Refer to Figures 13 and 14 for visual orientation. By triangulating $\mathcal{P}$, we obtain a triangulation T whose dual graph is a tree T'. We consider a path Π between two leaves (say, t_1 and t_k) in T', and obtain a *caterpillar* C' by adding as *feet* all vertices adjacent to Π; let C be the corresponding set of triangles (shown in dark cyan in Figure 13).

Now the idea is to place guards on vertices of C (that is a subset of the vertices of $\mathcal{P}$), aiming to see all of C. We then consider a recursive subdivision of $\mathcal{P}$ into caterpillars, by proceeding from foot triangles of covered caterpillars to ears, until all of $\mathcal{P}$ is covered; this corresponds to the colored subdivision in Figure 13.

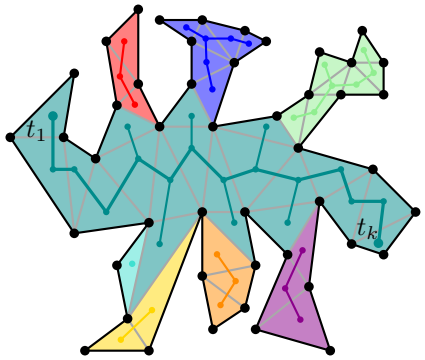


Figure 13: Polygon $\mathcal{P}$ in black, triangulation T in gray, and a partition into (colored) caterpillars.

To cover C, we start by placing a guard on a vertex v_0 of an ear triangle (say, t_1). If C' is a path (i.e., a caterpillar without foot triangles), we can proceed in a straightforward manner: Either the next triangles on the path are visible from the guard on v_0, or there is a reflex vertex v_r obstructing the view to a triangle t_i. In the latter case, we can place the next guard on an unseen vertex v_j of t_i, i.e., v_j is not seen by any of the previously placed guards; by assumption, the distance of v_0 and v_r is at least 1, as is the distance of v_r and v_j. Because v_r is reflex, a shortest path from v_0 to v_j has length at least 2 by triangle inequality.

This leaves the case in which we have foot triangles, which is analyzed in the following. Assume that we already placed a guard on a vertex incident to the path of the caterpillar. We argue how we proceed even if all

path triangles have incident foot triangles, that is, we show that we can place a set of guards that together monitor all caterpillar triangles, while ensuring a distance of at least 2 between any pair of guards. Furthermore, whenever we place a guard in a foot triangle, then this guard is never needed to cover any path triangles, hence, even if not all path triangles have incident foot triangles, we yield a feasible guard placement.

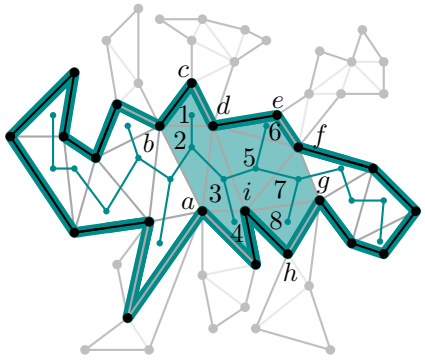


Figure 14: Vertices and triangular faces of a caterpillar for the proof of Theorem 5.

In the recursive call, we also take into account what previously placed guards see; note that unseen vertices are feasible guard locations with a distance of at least 2 to all previously placed guards.

To indicate that a vertex v is reflex in the polygonal chain u, v, w, we say that v *is reflex w.r.t.* $u - w$; note that the polygonal chain u, v, w must not be the polygon boundary. The line segment uv contained in $\mathcal{P}$ is denoted by $\overline{uv}$; it is either a diagonal or a polygon edge.

Now we consider the situation in Figure 14 and assume that a guard on a has been placed to monitor the triangle to the left of $\overline{ab}$. We aim to monitor triangles $1, 2, \ldots, 8$. The guard on a sees the triangles 2, 3 and 4. If a sees c, then a sees triangle 1 as well. If a does not see c, we place a guard on c (in this case either b or d is reflex w.r.t. $a - c$, thus, c has distance at least 2 to a).

We now provide a case distinction on the next placement(s) of guards. In case we placed a guard on c in addition to the guard on a, whenever we consider a seeing vertices, this also includes vertex c.

1. If i is reflex w.r.t. $a - g$, we place a guard on g; together these guards see triangles 5, 7, and 8.

- **(a)** If a or g see e, then they also see triangle 6.
- **(b)** Otherwise, we place another guard on e (which has distance of at least 2 to all guards placed before), which then monitors triangle 6.

2. Otherwise, i.e., i is not reflex w.r.t. $a - g$:

- **(a)** If d is reflex w.r.t. $a - e$:
  - **i.** If $\delta(a, f) \geq 2$, we place a guard on f to see triangles 5, 6, and 7. If h is seen by a or f, then also triangle 8 is seen. Otherwise, we place a guard on h to see triangle 8.
  - **ii.** Else if $\delta(a, g) \geq 2$, we place a guard on g, then a and g also see triangles 5, 7, and 8. If e is seen by a or g, then also triangle 6 is seen. Otherwise, we place a guard on e to see triangle 6.

In the remaining cases iii.– vi., we have $\delta(a, f) < 2$, $\delta(a, g) < 2$, thus, a sees both f and g.

  - **iii.** Else if e does not see either g or h (which implies $\delta(e, h) > 2$, $\delta(e, g) > 2$):
    If a does not see h, we place a guard on h, which covers triangle 8. Moreover, we also place a guard on e (which is neither seen from a or h), and the guards then also cover triangles 5, 6, and 7. Otherwise, i.e., a sees h, we place a guard on e to guarantee that triangles 5, 6, 7, and 8 are seen.
  - **iv.** Else if e sees g, but does not see h:
    If a does not see h, we place two guards on e and h, the guards together then guard triangles 5, 6, 7, and 8. Otherwise, a sees d, f, g, h, i and with that also triangles 5, 7, and 8; we place a guard on e, which sees triangle 6.
  - **v.** Else if e sees h, but does not see g:
    If $\delta(e, h) > 2$, we place a guard on each e and h, and thereby cover triangles 5, 6, 7, and 8. If $\delta(e, h) < 2$, Lemma 4 yields a contradiction to $\delta(a, g) < 2$ with $v_1 = e$, $v_2 = d$, $v_3 = a$, $v_4 = i$, $v_5 = h$, $v_6 = g$, and $v_7 = f$.
  - **vi.** Else if e sees g and h:
    If a sees h, we place a guard on e, and the guards then cover triangles 5, 6, 7, and 8. Otherwise, we place a guard on h, and if h sees f, triangles $5, \ldots, 8$ are seen. If not, we place a guard on e if $\delta(e, h) > 2$ and cover triangles $5, \ldots, 8$; otherwise, Lemma 4 yields a contradiction to $\delta(e, h) < 2$ with $v_1 = a, v_2 = i$, $v_3 = h, v_4 = g, v_5 = f, v_6 = e$, and $v_7 = d$.

- **(b)** Otherwise, a also sees f, hence, triangles 5, 6, and 7 are covered.
  - **i.** If a sees h, it also sees triangle 8.
  - **ii.** If a does not see h, we place a guard on h, which then sees triangle 8.

The guards we place in foot triangles are never needed to cover path triangles, hence, if some of the foot triangles did not exist, we can simply proceed along the caterpillar path (and place a guard there if a triangle is not (completely) seen). □

5 Conclusions and Future Work

We considered the Dispersive Art Gallery Problem with vertex guards, both in simple polygons and in polygons with holes, where we measure distance in terms of geodesics between any two vertices. We established NP-completeness of the problem of deciding whether there exists a vertex guard set with a dispersion distance of 2 for polygons with holes. For simple polygons, we presented a method for placing vertex guards with dispersion distance of at least 2. While we do not show NP-completeness of the problem in simple polygons, we conjecture the following.

Conjecture 1 *For a sufficiently large dispersion distance $\ell > 2$, it is* NP*-complete to decide whether a simple polygon allows a set of vertex guards with a dispersion distance of at least ℓ.*

Another open problem is to construct constant-factor approximation algorithms. This hinges on good lower bounds for the optimum.

Both our work and the paper by Rieck and Scheffer [14] consider vertex guards. This leaves the problem for point guards (with positions not necessarily at polygon vertices) wide open. Given that the classical AGP for point guards is $\exists\mathbb{R}$-complete [1], these may be significantly more difficult to resolve.

References

[1] M. Abrahamsen, A. Adamaszek, and T. Miltzow. The art gallery problem is $\exists\mathbb{R}$-complete. *Journal of the ACM*, 69(1):4:1–4:70, 2022.

[2] A. Bärtschi, S. K. Ghosh, M. Mihalák, T. Tschager, and P. Widmayer. Improved bounds for the conflict-free chromatic art gallery problem. In *Symposium on Computational Geometry (SoCG)*, pages 144–153, 2014.

[3] A. Bärtschi and S. Suri. Conflict-free chromatic art gallery coverage. *Algorithmica*, 68(1):265–283, 2014.

[4] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory*, 18(1):39–41, 1975.

[5] M. de Berg and A. Khosravi. Optimal binary space partitions for segments in the plane. *International Journal on Computational Geometry and Applications*, 22(3):187–206, 2012.

[6] L. H. Erickson and S. M. LaValle. A chromatic art gallery problem. Technical report, University of Illinois, 2010.

[7] S. P. Fekete, S. Friedrichs, M. Hemmer, J. S. B. Mitchell, and C. Schmidt. On the chromatic art gallery problem. In *Canadian Conference on Computational Geometry (CCCG)*, 2014.

[8] S. Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory*, 24(3):374, 1978.

[9] F. Hoffmann, K. Kriegel, S. Suri, K. Verbeek, and M. Willert. Tight bounds for conflict-free chromatic guarding of orthogonal art galleries. *Computational Geometry*, 73:24–34, 2018.

[10] C. Iwamoto and T. Ibusuki. Computational complexity of the chromatic art gallery problem for orthogonal polygons. In *Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 146–157, 2020.

[11] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[12] J. S. B. Mitchell. Private communication, 2018.

[13] J. O'Rourke. *Art gallery theorems and algorithms*. Oxford New York, NY, USA, 1987.

[14] C. Rieck and C. Scheffer. The dispersive art gallery problem. *Computational Geometry: Theory and Applications*, 117:102054, 2024.

[15] T. C. Shermer. Recent results in art galleries (geometry). *Proceedings of the IEEE*, 80(9):1384–1399, 1992.

[16] J. Urrutia. Art gallery and illumination problems. In *Handbook of Computational Geometry*, pages 973–1027, 2000.

Guarding Points on a Terrain by Watchtowers *

Byeonguk Kang† Junhyeok Choi‡ Jeesun Han† Hee-Kap Ahn§

Abstract

We study the problem of guarding points on an x-monotone polygonal chain, called a terrain, using k watchtowers. A watchtower is a vertical segment whose bottom endpoint lies on the terrain. A point on the terrain is visible from a watchtower if the line segment connecting the point and the top endpoint of the watchtower does not cross the terrain. Given a sequence of point sites lying on a terrain, we aim to partition the sequence into k contiguous subsequences and place k watchtowers on the terrain such that every point site in a subsequence is visible from the same watchtower and the maximum length of the watchtowers is minimized. We present efficient algorithms for two variants of the problem.

1 Introduction

A *terrain* is a graph of a piecewise linear function $f : A \subset \mathbb{R} \to \mathbb{R}$ that assigns a height $f(p)$ to every point p in the domain A of the terrain. In other words, a terrain is an x-monotone polygonal chain in the plane. A *watchtower* is a vertical segment whose bottom endpoint lies on the terrain. A point on the terrain is *visible* from a watchtower if the line segment connecting the point and the top endpoint of the watchtower does not cross the terrain. If a point is visible from a watchtower, we say that the point is *guarded* by the watchtower. We say that a set of points is guarded by a watchtower if every point in the set is guarded by the watchtower.

In this paper, we study the following problem of guarding point sites on a terrain using k watchtowers: Given a sequence of point sites on a terrain, partition it into k subsequences and place k watchtowers on the terrain such that every point site in a subsequence is guarded by the same watchtower and the maximum length of the watchtowers is minimized. We call it the *contiguous k-watchtower problem* for point sites on a terrain. We also consider the problem with an additional condition on the placement of watchtowers: a watchtower guarding a subsequence of point sites must be placed in the x-range $x_{\min} \leqslant x_w \leqslant x_{\max}$ of the point sites in the subsequence, where x_w is the x-coordinate of the watchtower and $x_{\min}$ (resp. $x_{\max}$) is the minimum (resp. maximum) x-coordinates of the point sites in the subsequence. This is the *in-place* version of the contiguous k-watchtower problem for point sites on a terrain. For both problems, we call those k watchtowers satisfying the conditions and minimizing the maximum length the *optimal* k watchtowers. See Figure 1 for an illustration for the problems.

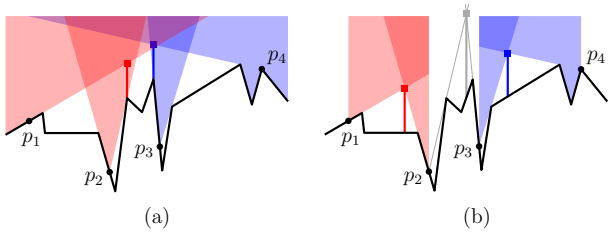


Figure 1: (a) Optimal watchtowers for the contiguous 2-watchtower problem. The red tower guards p_1 and p_2, and the blue tower guards p_3 and p_4. (b) Optimal watchtowers for the in-place version. The red watchtower guards p_1 and p_2, and it is placed in the x-range of p_1 and p_2. The blue watchtower guards p_3 and p_4, and it is placed in the x-range of p_3 and p_4. To guard point sites including both p_2 and p_3 using one watchtower, the watchtower must be at least as long as the gray watchtower.

The k-watchtower problems we consider have applications in several domains, including geographic information system, communication tower locations, and military surveillance [4].

1.1 Related works

A fair amount of work has been done on minimizing the number of guards in various settings. The art gallery problem [10] asks for the minimum number of point

*This research was supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (RS-2019-II191906, Artificial Intelligence Graduate School Program(POSTECH)). This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (RS-2023-00219980).

†Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. Email: {kbu417, 6627hjs, heekap}@postech.ac.kr

‡Bagelcode, Seoul, Korea. Email: cjh0630@postech.ac.kr

§Graduate School of Artificial Intelligence, Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. Email: heekap@postech.ac.kr

guards that together guard the whole art gallery, represented by a simple polygon. The art gallery problem was first posed by Klee in 1973 [10]. Chvátal and Fisk [5, 8] gave an upper bound $\lfloor n/3 \rfloor$ on the minimum number of point guards for a simple polygon with n vertices.

The terrain guarding problem [9] asks for the minimum number of point guards lying on the terrain that together guard the terrain. Cole and Sharir [6] showed that finding the minimum number of guards for a polyhedral terrain in 3-dimensional space is NP-complete. Later, Chen et al. [3] showed that the same problem for a terrain in 2-dimensional space is also NP-complete.

The k-watchtower problem for a terrain with n vertices in 2-dimensional space is to minimize the maximum length of k watchtowers that together guard the whole terrain. The 2-watchtower problem was first studied by Bespamyatnikh et al. [2]. They presented an $O(n^3 \log^2 n)$-time algorithm for the variant, called the discrete version, in which every watchtower must be placed at a vertex of the terrain. They also gave an $O(n^4 \log^2 n)$-time algorithm for the continuous version in which the two watchtowers can be placed anywhere in the terrain. Agarwal et al. [1] improved the results by an $O(n^2 \log^4 n)$-time algorithm for the discrete version and by an $O(n^3 \alpha(n) \log^3 n)$-time algorithm for the continuous version.

There are also a few results for the k-watchtower problem for a 2-dimensional terrain with n vertices in 3-dimensional space. Agarwal et al. [1] presented an $O(n^{11/3}\text{polylog}(n))$-time algorithm for the discrete version of the 2-watchtower problem. Recently, Tripathi et al. [12] gave an algorithm for the discrete version of the k-watchtower problem that runs in $O(n^{k+3}k^2\alpha^2(n)\log^2 n + n^7\alpha^3(n)\log n)$ time.

To the best of our knowledge, little is known about guarding a finite set of input points lying on a terrain, not the whole terrain, except the one by Agarwal et al. [1]. They considered the 2-watchtower problem for guarding a finite set of m point sites on a terrain with n vertices in 2-dimensional space where every point site must be guarded by at least one of the two watchtowers. The watchtowers can be placed anywhere in the terrain. They presented an $O(mn \log^4 n)$-time algorithm for the problem. One may wonder if this algorithm extends to the k-watchtower problem for $k \geqslant 3$. It seems to us that it does, but the running time becomes exponential in k for m point sites lying on a terrain with n vertices.

1.1.1 Our results.

We consider the contiguous k-watchtower problem and the in-place contiguous k-watchtower problem for m point sites lying on a terrain with n vertices in the plane. For ease of the description, we may call the in-place contiguous k-watchtower problem the in-place k-watchtower problem. If $k \geqslant m$ (resp. $k \geqslant n$), we place one watchtower with zero length on every point site (resp. on every vertex of the terrain). Considering the cost of watchtowers, it is desirable to use a small number of watchtowers for point sites. Therefore, we assume that $k \ll \min\{n, m\}$.

For $k = 1$, we present an algorithm with running time $O(m+n)$ for both problems. Observe that the running time is linear to the complexity of the input. This is an improvement upon the previously best algorithm with running time $O(mn)$ [1].

For the contiguous k-watchtower problem, the watchtowers can be placed anywhere in the terrain. We show a monotonicity on the minimum length of a watchtower, and present an $O((m+n)\log m)$-time algorithm for $k = 2$. For $k \geqslant 3$, we can solve the problem in $O(k(n+m)\log^{\lceil \log_2 k \rceil} m)$ time. Our algorithm runs in $O((m+n)\log^{\lceil \log_2 k \rceil} m)$ time for any fixed k.

For the in-place k-watchtower problem, a watchtower guarding a contiguous subsequence of point sites must be placed in the x-range of the subsequence. We observe that the monotonicity shown for the contiguous k-watchtower problem does not hold for this problem. We present an $O((m+n)\log(m+n))$-time algorithm for $k = 2$ and an $O(km^2 + (mn+m^2)\log(m+n))$-time algorithm for $k \geqslant 3$. Our algorithm runs in $O((mn+m^2)\log(m+n))$ time for any fixed $k \geqslant 3$.

1.1.2 Sketch of our algorithms.

We devise an efficient algorithm for the contiguous k-watchtower problem for $k = 1$ that runs in $O(m+n)$ time. The *visibility region* of a point site is the set of points visible from the point site. To find an optimal watchtower, we need to compute the intersection of the visibility regions of point sites. The previous algorithm takes $O(mn)$ time in computing visibility regions of point sites and their intersection [1]. To do this efficiently, we define a region $W(p,q)$ for a pair of point sites (p,q) such that $W(p,q)$ contains the intersection of the visibility regions of p and q. We show that the intersection of visibility regions of all point sites can be computed in $O(m+n)$ time using the intersection of $W(p,q)$'s for all pairs of point sites (p,q). From this, we can compute an optimal watchtower for m point sites lying on a terrain with n vertices in $O(m+n)$ time.

For $k \geqslant 2$, we show a monotonicity stating that the length of an optimal watchtower for a subsequence P_1 of point sites is at least the length of an optimal watchtower for any subsequence of P_1. Based on the monotonicity, our algorithm for the contiguous k-watchtower problem uses binary search to find an optimal partition of the point site set that minimizes the maximum length of the watchtowers. In each step of the binary search, we partition the point site sequence into two contiguous subsequences. Then, we compute the optimal length of the watchtowers for each subsequence using half of the

watchtowers. If the optimal length of the watchtowers for the left subsequence is larger than the right subsequence's, then we find an optimal partition index in the left half of indices of the point site set. When the number of the watchtower is one, we can compute the optimal length of the half of the watchtowers in $O(m+n)$ time by using the algorithm for one watchtower.

In the in-place k-watchtower problem, the monotonicity used for our algorithm for the contiguous k-watchtower problem does not hold. So we consider every possible partition of the sequence into k contiguous subsequences. For $k=2$, there are $O(m)$ different partitions. A naïve approach is to compute the optimal tower-length for every partition in $O(m^2+mn)$ total time by applying the algorithm for the contiguous 1-watchtower problem. We compute optimal watchtowers efficiently as follows. For every prefix of the input sequence of point sites, we compute the intersection of $W(p,q)$'s for every pair of point sites (p,q) in the prefix. We compute those intersections incrementally in the length of the prefixes in $O((m+n)\log(m+n))$ total time. Using those intersections, we can compute optimal two watchtowers in $O((m+n)\log(m+n))$ time.

For $k \geqslant 3$, a naïve approach is to consider $O(m^{k-1})$ different partitions, compute their optimal tower-lengths, and then return the minimum one among them. To compute optimal k watchtowers efficiently, we compute the minimum length of one watchtower for every contiguous subsequence incrementally in $O((m^2+mn)\log(m+n))$ total time in the preprocessing. Then we find an optimal partition by dynamic programming that has $O(km^2)$ subproblems.

Most proofs are omitted and they will be given in a full version.

2 Preliminaries

For a point p in the plane, we use $x(p)$ and $y(p)$ to denote the x- and y-coordinates of p. For two distinct points p and q in the plane, let pq denote the line segment connecting p and q, and let $\overline{pq}$ denote the line passing through both p and q. For a nonvertical line L, we use L^+ to denote the set of points in $\mathbb{R}^2$ that lie on or above L, and L^- to denote the set of points in $\mathbb{R}^2$ that lie on or below L.

A region A is *x-monotone* if for every line L perpendicular to the x-axis, $A \cap L$ is connected. A region A is *unbounded vertically upwards* if any vertically upward ray emanating from a point in A is contained in A. A polygonal chain B is *x-monotone* if for every line L perpendicular to the x-axis, either $B \cap L = \emptyset$ or it is a point. We use $T = \langle v_1, \ldots, v_n \rangle$, a sequence of vertices with $x(v_i) < x(v_j)$ for any $1 \leqslant i < j \leqslant n$, to denote an x-monotone polygonal chain which we call a *terrain* in 2-dimensional space. Without loss of generality, we assume $n \geqslant 2$. For any two points $p, q \in T$ with $x(p) \leqslant x(q)$, let $T(p,q)$ denote the subchain of T from p to q, and let $T^+(p,q)$ denote the set of points $z \in \mathbb{R}^2$ such that $x(p) \leqslant x(z) \leqslant x(q)$ and $y(z) \geqslant y(z')$, where z' is a point in T with $x(z) = x(z')$. We simply use T^+ to denote $T^+(v_1, v_n)$. We denote by $P = \langle p_1, \ldots, p_m \rangle$ a sequence of m point sites lying on T such that $x(p_i) < x(p_j)$ for $1 \leqslant i < j \leqslant m$. We denote by $P(i,j)$ the contiguous subsequence $\langle p_i, \ldots, p_j \rangle$ of P for $1 \leqslant i < j \leqslant m$. For ease of description, we assume that $m \geqslant 2$, and let $p_0 = v_1$ and $p_{m+1} = v_n$. We use $T(i,j)$ to denote $T(p_i, p_j)$, and $T^+(i,j)$ to denote $T^+(p_i, p_j)$.

A point $p \in \mathbb{R}^2$ is *visible* from a point $q \in \mathbb{R}^2$ if and only if pq is contained in T^+. For a point $q \in T$, let $V(q)$ denote the *visibility region* of q, which consists of the points in T^+ visible from q. For a point site $p_i \in P$, we use $V(i)$ to denote $V(p_i)$. Observe that $V(i)$ is connected and unbounded vertically upwards. Let $\mathbb{V}(i,j) = \bigcap_{i \leqslant \ell \leqslant j} V(p_\ell)$. The following observation is straightforward.

Observation 1 *The point sites in $P(i,j)$ are visible from a watchtower if and only if the top endpoint of the watchtower is contained in $\mathbb{V}(i,j)$.*

For any two real values a, b with $a \leqslant b$, we use $S(a,b)$ to denote the vertical slab between the lines $x = a$ and $x = b$. In other words, it is the set of points $z \in \mathbb{R}^2$ such that $a \leqslant x(z) \leqslant b$. For any two points $p, q \in \mathbb{R}^2$ with $x(p) \leqslant x(q)$, we abuse the notation so that $S(p,q)$ denotes $S(x(p), x(q))$. We use $S(i,j)$ to denote $S(p_i, p_j)$. For a set $A \subset \mathbb{R}^2$, we use $S(A)$ to denote the smallest vertical slab containing A.

For any two sets A and B of points, let $d_y(A,B)$ denote the minimum vertical distance between A and B, that is, $d_y(A,B) = \min_{p_A \in A, p_B \in B} |y(p_A) - y(p_B)|$ subject to $x(p_A) = x(p_B)$. If there are no two points $p_A \in A$ and $p_B \in B$ with $x(p_A) = x(p_B)$, we set $d_y(A,B) = \infty$. We say that A lies *left* to B if the rightmost point p of A and the leftmost point of q of B satisfy $x(p) \leqslant x(q)$.

3 Contiguous k watchtowers

In this section, we present an $O(k(n+m)\log^{\lceil \log_2 k \rceil} m)$-time algorithm for the contiguous k-watchtower problem for point sites P on a terrain T. In Section 3.1, we present an $O(m+n)$-time algorithm for computing an optimal watchtower for $P = \langle p_1, \ldots, p_m \rangle$. We use the algorithm for one watchtower together with binary search in computing the optimal k watchtowers for $k \geqslant 2$ in Sections 3.2 and 3.3. For any constant k, the algorithm runs in near-linear time: $O((m+n)\log m)$ time for $k=2$, and $O((m+n)\log^{\lceil \log_2 k \rceil} m)$ time for any fixed k.

3.1 An optimal watchtower for a site sequence

We consider the problem of placing a shortest watchtower that guards all point sites of P. Let $F(1,m)$ denote the minimum length of a watchtower that guards all point sites in P. By Observation 1, any watchtower guarding point sites in P must have its top endpoint contained in $\mathbb{V}(1,m)$. Thus, $F(1,m) = d_y(T, \mathbb{V}(1,m))$.

A straightforward way to compute an optimal watchtower for the sequence is to compute $V(\ell)$ for all $\ell = 1,\ldots,m$, compute their intersection $\mathbb{V}(1,m)$, and then compute $F(1,m)$. Observe that it already takes $O(mn)$ time for computing $V(\ell)$ for all $\ell = 1,\ldots,m$ [11].

We show how to compute $\mathbb{V}(1,m) = \bigcap_{1\leqslant\ell\leqslant m} V(\ell)$ efficiently, in $O(m+n)$ time. Before showing this, we need to define a region $R(1,m)$ for $P(1,m)$. Let L be line $\overline{p_1p_m}$ if p_m is visible from p_1. If p_m is not visible from p_1, let L be line $\overline{uv}$, where uv is the edge of $V(1)$ with $x(u) < x(p_m) \leqslant x(v)$. If p_m lies on a vertex of T, let $R(1,m)$ be the set of points $z \in L^+$ satisfying $x(z) \geqslant x(p_m)$. If p_m is contained in the interior of an edge e of T, let $R(1,m)$ be the set of points $z \in L^+ \cap \overline{e}^+$ satisfying $x(z) \geqslant x(p_m)$. See Figure 2 for an illustration for four possible cases. We define the region $R(m,1)$ symmetrically.

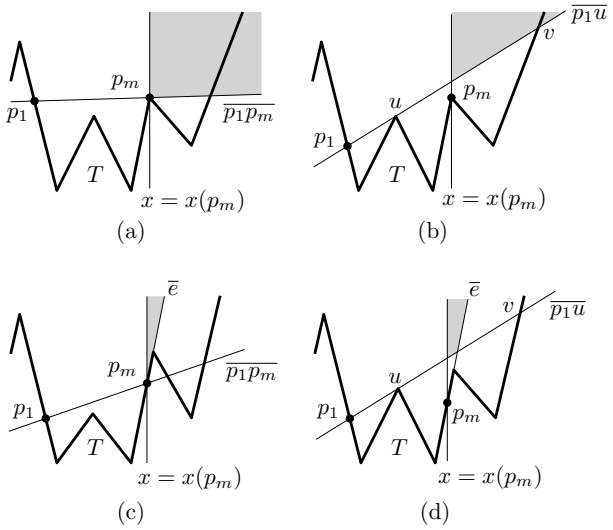


Figure 2: $R(1,m)$ in gray region. (a) p_m lying on a vertex of T and visible from p_1. (b) p_m lying on a vertex of T and not visible from p_1. (c) p_m lying in the interior of an edge e of T and visible from p_1. (d) p_m lying in the interior of an edge e of T and not visible from p_1.

By definition, $R(1,m)$ is the intersection of two or three closed half-planes. Thus, $R(1,m)$ is convex. Moreover, it is unbounded vertically upwards.

Combining $R(1,m), R(m,1)$, and $V(1) \cap V(m)$ restricted to $S(1,m)$, we define $W(1,m)$ as follows. See Figure 3 for an illustration.

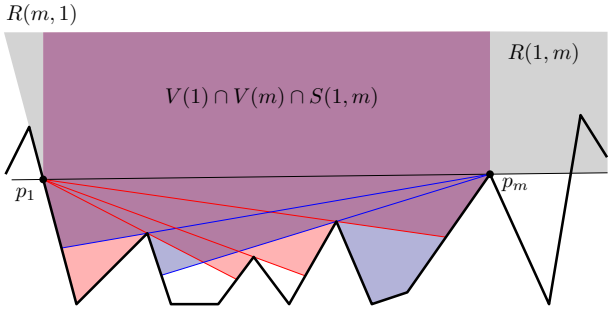


Figure 3: The purple region is $V(1) \cap V(m) \cap S(1,m)$. $W(1,m)$ is the union of the purple region and the right gray region from $R(1,m)$ and the left gray region from $R(m,1)$.

$$W(1,m) = R(1,m) \cup R(m,1) \cup \big(V(1) \cap V(m) \cap S(1,m)\big).$$

By definition, $W(1,m)$ is connected and unbounded vertically upwards.

Observation 2 *The followings hold by the definition of* $W(1,m)$.

(a) $W(1,m) \cap S(1,m) = V(1) \cap V(m) \cap S(1,m)$.
(b) $W(1,m) \cap S(0,1) = R(m,1) \cap S(0,1)$.
(c) $W(1,m) \cap S(m,m+1) = R(1,m) \cap S(m,m+1)$.
(d) *For* $q \in \{p_1, p_m\}$, $y(q) \leqslant y(z)$ *for all* $z \in W(1,m)$ *with* $x(z) = x(q)$.

Based on Observation 2, we can compute $W(1,m)$ efficiently.

Lemma 1 *We can compute* $W(1,m)$ *in time linear to the complexity of* $T(1,m)$.

By Lemma 1, $W(\ell,\ell+1)$ can be computed in time linear to the complexity of $T(\ell,\ell+1)$. Thus, we can compute $W(\ell,\ell+1)$ for all $\ell = 1,\ldots,m$ in $O(m+n)$ time. We show a few properties useful for computing $\mathbb{V}(1,m)$ efficiently.

Lemma 2 $\mathbb{V}(1,m) = \bigcap_{1<\ell\leqslant m} W(\ell-1,\ell) \cap V(1) \cap V(m)$.

Let $X_1 = \bigcap_{1<\ell\leqslant r} R(\ell-1,\ell)$, $X_2 = \bigcap_{r<\ell\leqslant m} R(\ell,\ell-1)$, $X_3 = V(r) \cap V(r+1) \cap V(1) \cap V(m)$, and $X_4 = S(r,r+1)$. By Lemma 2 and Observation 2(c),

$$\mathbb{V}(1,m) \cap S(r,r+1) = X_1 \cap X_2 \cap X_3 \cap X_4. \qquad (1)$$

We need the following lemma to show that $\mathbb{V}(1,m)$ can be computed in $O(m+n)$ time.

Lemma 3 *We can compute* $\bigcap_{1<\ell\leqslant r} R(\ell-1,\ell) \cap S(r,r+1)$ *for all* $r = 2,\ldots,m$ *in* $O(m+n)$ *time.*

Theorem 4 *We can compute a minimum-length watchtower that guards m point sites lying on an x-monotone polygonal chain with n vertices in $O(m+n)$ time.*

Proof. Note that $F(1,m) = d_y(T, \mathbb{V}(1,m))$. First, we show how to compute $\mathbb{V}(1,m)$ in $O(m+n)$ time. We can get $\mathbb{V}(1,m)$ by gluing $\mathbb{V}(1,m) \cap S(0,2)$, $\mathbb{V}(1,m) \cap S(2,m-1)$, and $\mathbb{V}(1,m) \cap S(m-1,m+1)$.

We compute $\mathbb{V}(1,m) \cap S(r,r+1)$ for all $r = 2,\ldots,m-1$ which is defined in Equation 1. By Lemma 3, we can compute $\bigcap_{1<\ell\leqslant r} R(\ell-1,\ell) \cap S(r,r+1)$ for all $r = 2,\ldots,m-1$ in $O(m+n)$ time. Their total complexity is $O(m+n)$. Similarly, we can compute $\bigcap_{r<\ell\leqslant m} R(\ell,\ell-1) \cap S(r,r+1)$ for all $r = 2,\ldots,m-1$ in $O(m+n)$ time. Their total complexity is $O(m+n)$. By Lemma 1, we can compute $V(r) \cap V(r+1) \cap S(r,r+1)$ for all $r = 2,\ldots,m-1$ in $O(m+n)$ time. Their total complexity is $O(m+n)$. Recall that we can compute $V(1)$ and $V(m)$ in $O(n)$ time [11]. Observe that every region that we compute is x-monotone. Thus, we can compute the intersections $\mathbb{V}(1,m) \cap S(r,r+1)$ of those regions for all $r = 2,\ldots,m-1$ in time linear to their total complexity $O(m+n)$ by linear scan. Similarly, $\mathbb{V}(1,m) \cap S(0,2)$ and $\mathbb{V}(1,m) \cap S(m-1,m+1)$ can be computed in $O(m+n)$ time.

We glue $\mathbb{V}(1,m) \cap S(0,2)$, $\mathbb{V}(1,m) \cap S(2,m-1)$, and $\mathbb{V}(1,m) \cap S(m-1,m+1)$ together and get $\mathbb{V}(1,m)$. Since the complexity of $\mathbb{V}(1,m)$ is $O(m+n)$, we can compute $F(1,m) = d_y(T, \mathbb{V}(1,m))$ in $O(m+n)$ time by linear scan. We compute the location of an optimal watchtower during the scan. □

3.2 Two watchtowers

We consider the contiguous k-watchtower problem for $k = 2$: Partition P into 2 subsequences and place 2 watchtowers on T such that every point site in a subsequence is guarded by the same watchtower and the maximum length of the watchtowers is minimized.

Recall that $P(i,j)$ denotes the contiguous subsequence $\langle p_i,\ldots,p_j\rangle$ of $P = \langle p_1,\ldots,p_m\rangle$ for $1 \leqslant i < j \leqslant m$. Let $F(i,j)$ denote the minimum length of a watchtower that guards point sites in $P(i,j)$ lying on T. We have the following lemma stating the monotonicity on $F(i,j)$ obtained by $\mathbb{V}(i',j') \subseteq \mathbb{V}(i,j)$.

Lemma 5 *For indices i', i, j and j' satisfying $1 \leqslant i' \leqslant i \leqslant j \leqslant j' \leqslant m$, $F(i,j) \leqslant F(i',j')$.*

For an index i with $1 \leqslant i < m$, let $F_1(i) = F(1,i)$ and $F_2(i) = F(i+1,m)$. Then the minimum length for two watchtowers is $\min_{1\leqslant i<m}\{\max\{F_1(i),F_2(i)\}\}$. By Lemma 5, $F_1(i)$ increases monotonically and $F_2(i)$ decreases monotonically as i increases from 1 to $m-1$. Therefore, we find the index that achieves the minimum length by binary search. Since P consists of m point sites, the number of binary search steps is $O(\log m)$. By Theorem 4, the comparison in each step can be done in $O(m+n)$ time. In other words, we can compute both $F_1(i)$ and $F_2(i)$ for any index $i = 1,\ldots,m-1$ in $O(m+n)$ time. Also, we can compute the location of an optimal watchtower for $P(i,j)$ for any index $1 \leqslant i \leqslant j \leqslant m$ in $O(m+n)$ time by Theorem 4. Therefore, we can compute the optimal two watchtowers in $O((m+n)\log m)$ time.

Theorem 6 *We can compute optimal two watchtowers for the contiguous 2-watchtower problem with m point sites lying on an x-monotone polygonal chain with n vertices in $O((m+n)\log m)$ time.*

3.3 k watchtowers

In this section, we present an $O(k(n+m)\log^{\lceil\log_2 k\rceil} m)$-time algorithm for computing the contiguous k watchtowers of minimum length for $k \geqslant 3$. Roughly speaking, we partition P into two contiguous subsequences and compute the minimum tower-length for one subsequence using $\lfloor k/2\rfloor$ watchtowers and the minimum tower-length for the other subsequence using $\lceil k/2\rceil$ watchtowers. We repeat this recursively.

For indices $1 \leqslant i \leqslant j \leqslant m$, let $\mathsf{opt}(i,j,k')$ denote the minimum tower-length for $P(i,j)$ using k' watchtowers with $k' \geqslant 1$. Obviously, $\mathsf{opt}(i,j,k') \geqslant \mathsf{opt}(i,j,k'+1)$. Observe that $\mathsf{opt}(i,j,1) = F(i,j)$. For $k' \geqslant 2$, $\mathsf{opt}(i,j,k')$ equals to

$$\min_{i\leqslant\ell<j}\{\max\{\mathsf{opt}(i,\ell,\lfloor k'/2\rfloor),\mathsf{opt}(\ell+1,j,\lceil k'/2\rceil)\}\}.$$

Lemma 7 *$opt(1,i,k') \leqslant opt(1,j,k')$ for $1 \leqslant i \leqslant j \leqslant m$ and $k' \geqslant 1$.*

The minimum tower-length for $P(1,m)$ using k watchtowers is $\mathsf{opt}(1,m,k)$. By Lemma 7, we can find an index $\ell = \arg\min_{1\leqslant\ell<m}\max\{\mathsf{opt}(1,\ell,\lfloor k'/2\rfloor),\mathsf{opt}(\ell+1,m,\lceil k'/2\rceil)\}$ by binary search. Therefore, we conclude this section with Theorem 8.

Theorem 8 *We can compute optimal k watchtowers for the contiguous k-watchtower problem with m point sites lying on an x-monotone polygonal chain with n vertices in $O(k(n+m)\log^{\lceil\log_2 k\rceil} m)$ time.*

4 In-place contiguous k watchtowers

In this section, we present algorithms for the in-place k-watchtower problem for P lying on T. In this problem, a watchtower that guards a subsequence $P(i,j)$ must be placed in $T(i,j)$. By the problem definition, no watchtower cannot be placed on $T(0,1) \cup T(m,m+1)$. Thus, for ease of discussion, we assume that p_1 lies on v_1 and p_m lies on v_n.

In Section 4.1, we present an $O((m+n)\log(m+n))$-time algorithm for $k = 2$. The algorithm works in incremental fashion in computing an optimal solution using a balanced binary search tree based on the segment tree [7]. In Section 4.2, we present an $O(km^2 + (mn+m^2)\log(m+n))$-time algorithm for $k \geqslant 3$. The algorithm uses dynamic programming in computing an optimal solution, using the $O((m+n)\log(m+n))$-time algorithm for $k = 2$ for the base case.

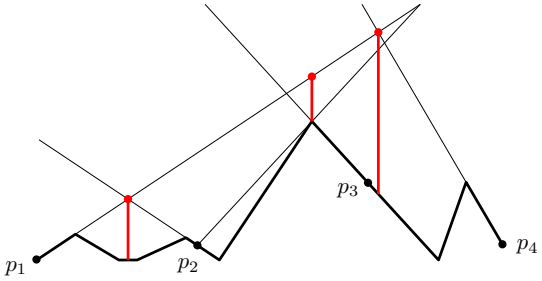


Figure 4: The vertical red line segments, left to right, are the shortest watchtowers for $P(1,2)$, $P(1,3)$, and $P(1,4)$. We have $F_1(2) > F_1(3)$ and $F_1(3) < F_1(4)$.

4.1 Two watchtowers

Let $F(i,j)$ denote the minimum length of one watchtower placed on $T(i,j)$ for $P(i,j)$ with $1 \leqslant i \leqslant j \leqslant m$. Let $F_1(i) = F(1,i)$ and $F_2(i) = F(i+1,m)$. Then our goal is to compute $\min_{1\leqslant i<m}\{\max\{F_1(i), F_2(i)\}\}$.

Observe that the monotonicity in Lemma 5 does not hold for the in-place k-watchtower problem due to the in-place requirement. For two indices i, j with $1 \leqslant i < j \leqslant m$, the watchtower for $P(1,j)$ can be placed anywhere in $T(1,j) = T(1,i) \cup T(i,j)$ while the watchtower for $P(1,i)$ must be placed in $T(1,i)$. So it is possible that $F_1(i) > F_1(j)$. See Figure 4.

We use an incremental algorithm for computing $F_1(i)$ and $F_2(i)$ for all $i = 1, \ldots, m-1$ that runs in $O((m+n)\log(m+n))$ time. Recall that we can compute $W(i-1,i)$ for all $i = 2, \ldots, m$ in $O(m+n)$ time by Lemma 1. Thus, we compute their intersection incrementally.

Let $\mathbb{W}(i) = \bigcap_{1<\ell\leqslant i} W(\ell-1,\ell)$. Recall that $W(\ell-1,\ell)$ is connected and unbounded vertically upwards. Thus, $\mathbb{W}(i)$ is connected and unbounded vertically upwards.

Lemma 9 *$\mathbb{V}(1,i) \cap S(1,i) = \mathbb{W}(i) \cap S(1,i)$.*

Corollary 10 *$d_y(T(1,i), \mathbb{V}(1,i)) = d_y(T(1,i), \mathbb{W}(i))$.*

By Observation 1, Lemma 9, and Corollary 10, $F_1(i) = d_y(T(1,i), \mathbb{W}(i))$. Our algorithm starts with trivial base case $F_1(1) = 0$ and computes $F_1(i)$ for all $i = 2, \ldots, m-1$ one by one incrementally.

First, we show that $\mathbb{W}(i)$ for all $i = 2, \ldots, m$ can be computed in $O((m+n)\log(m+n))$ time in total. We can compute $\mathbb{W}(2) = W(1,2)$ in $O(m+n)$ time. We show how to compute $\mathbb{W}(i+1) = \mathbb{W}(i) \cap W(i,i+1)$ from $\mathbb{W}(i)$ efficiently. To do this, we show that the boundary of $W(i,i+1)$ intersects the boundary of $\mathbb{W}(i)$ in $O(|T(i,i+1)|)$ connected components. In specific, each edge of $W(i,i+1)$ intersects the boundary of $\mathbb{W}(i)$ at most twice.

Lemma 11 *We can compute $F_1(i)$ and $F_2(i)$ for all $i = 1, \ldots, m-1$ in $O((m+n)\log(m+n))$ time.*

Recall that the minimum tower-length is $\min_{1\leqslant i<m}\{\max\{F_1(i), F_2(i)\}\}$. By Lemma 11, we can compute $F_1(i)$ and $F_2(i)$ in $O((m+n)\log(m+n))$ time for all $i = 1, \ldots, m-1$. Then, we can find $\min_{1\leqslant i<m}\{\max\{F_1(i), F_2(i)\}\}$ in $O(m)$ time. Recall that we can compute an optimal watchtower that guards $P(i,j)$ in $O(m+n)$ time by Theorem 4. In conclusion, we can compute the minimum tower-length and the locations of the optimal watchtowers in $O((m+n)\log(m+n))$ time.

Theorem 12 *We can compute optimal two watchtowers for the in-place contiguous 2-watchtower problem with m point sites lying on an x-monotone polygonal chain with n vertices in $O((m+n)\log(m+n))$ time.*

4.2 k watchtowers

Now we consider the in-place contiguous k watchtower problem for $k \geqslant 3$. By the definition of the problem, the minimum tower-length is

$$\min_{1\leqslant i_1<\ldots<i_{k-1}<m}\{\max\{F(1,i_1), \ldots, F(i_{k-1}+1, m)\}\}.$$

A naïve approach is to consider all combinations of $k-1$ point sites with indices $1 \leqslant i_1 < \ldots < i_{k-1} < m$ among m point sites, compute their maximum tower-lengths $\max\{F(1,i_1), F(i_1+1,i_2), \ldots, F(i_{k-1}+1,m)\}$, and then return the minimum one among the tower-lengths. This takes $O(m^{k-1}(m+n))$ time.

We can improve the running time using dynamic programming as follows. For an index $1 \leqslant i \leqslant m$, let $\mathsf{opt}(i,k')$ denote the minimum tower-length for the in-place k'-watchtower problem for $P(1,i)$. Then (1) $\mathsf{opt}(i,1) = F(1,i)$, (2) $\mathsf{opt}(i,k') = 0$ if $k' > 1$ and $i \leqslant k'$, and (3) $\mathsf{opt}(i,k') = \min_{1\leqslant j<i}\{\max\{\mathsf{opt}(j,k'-1), F(j+1,i)\}\}$ if $k' > 1$ and $i > k'$.

The optimal length is $\mathsf{opt}(m,k)$ and the number of subproblems is $O(km^2)$. To obtain $\mathsf{opt}(m,k)$, we need to compute $F(i,j)$ for all $1 \leqslant i \leqslant j \leqslant m$. By Theorem 12, for a fixed index $1 \leqslant i \leqslant m$, we can compute $F(i,j)$ for all $i \leqslant j \leqslant m$ in $O((m+n)\log(m+n))$ time. Therefore, we have the following lemma.

Lemma 13 *We can compute $F(i,j)$ for every $1 \leqslant i \leqslant j \leqslant m$ in $O((mn+m^2)\log(m+n))$ time.*

After $O((mn+m^2)\log(m+n))$-time preprocessing by Lemma 13, we can compute the minimum tower-length in $O(km^2)$ time using dynamic programming.

Theorem 14 *We can compute optimal k watchtowers for the in-place contiguous k-watchtower problem with m point sites lying on an x-monotone polygonal chain with n vertices in $O(km^2+(mn+m^2)\log(m+n))$ time.*

We would like to mention that the algorithm presented in this paper also work with little modification and without increasing the running time for minimizing the sum of the tower-lengths for k watchtowers.

References

[1] P. K. Agarwal, S. Bereg, O. Daescu, H. Kaplan, S. Ntafos, M. Sharir, and B. Zhu. Guarding a terrain by two watchtowers. *Algorithmica*, 58(2):352–390, 2010.

[2] S. Bespamyatnikh, Z. Chen, K. Wang, and B. Zhu. On the planar two-watchtower problem. In *Computing and Combinatorics: 7th Annual International Conference (COCOON 2001)*, pages 121–130, 2001.

[3] D. Z. Chen, V. Estivill-Castro, and J. Urrutia. Optimal guarding of polygons and monotone chains. In *7th Canadian Conference on Computational Geometry (CCCG 1995)*, pages 133–138, 1995.

[4] R. L. Church. Geographical information systems and location science. *Computers & Operations Research*, 29(6):541–562, 2002.

[5] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[6] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *Journal of symbolic Computation*, 7(1):11–30, 1989.

[7] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Spinger, 2008.

[8] S. Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374, 1978.

[9] J. King and E. Krohn. Terrain guarding is NP-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011.

[10] J. O'rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

[11] R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[12] N. Tripathi, M. Pal, M. De, G. Das, and S. C. Nandy. Guarding polyhedral terrain by k-watchtowers. In *Frontiers in Algorithmics: 12th International Workshop (FAW 2018)*, pages 112–125, 2018.

Multirobot Watchman Routes in a Simple Polygon

Joseph S. B. Mitchell* Linh Nguyen†

Abstract

The well-known Watchman Route problem seeks a shortest route in a polygonal domain from which every point of the domain can be seen. In this paper, we study the cooperative variant of the problem, namely the k-Watchmen Routes problem, in a simple polygon P. We look at both the version in which the k watchmen must collectively see all of P, and the quota version in which they must see a predetermined fraction of P's area.

We give an exact pseudopolynomial time algorithm for the k-Watchmen Routes problem in a simple orthogonal polygon P with the constraint that watchmen must move on axis-parallel segments, and there is a given common starting point on the boundary. Further, we give a fully polynomial-time approximation scheme and a constant-factor approximation for unconstrained movement. For the quota version, we give a constant-factor approximation in a simple polygon, utilizing the solution to the (single) Quota Watchman Route problem.

1 Introduction

In 1973, Victor Klee introduced the Art Gallery problem: given an art gallery with n walls (a polygon P), determine the minimum number of stationary guards at points within P such that every point of P can be seen by at least one guard point. The Art Gallery problem and its many variants have since been the subject of a large body of research in computational geometry and algorithms.

When guards are mobile, a single guard suffices to see a connected domain; thus, we are interested in finding routes for one or more guards that optimize some aspects of the guard(s)' movement (e.g., path lengths, the number of turns, etc). The problem of minimizing the distance that one guard must travel to see the entire polygon is the Watchman Route problem (WRP). Chin and Ntafos [3] introduced the WRP, proved NP-hardness in polygons with holes (see [6]) and gave an $O(n)$ algorithm for simple orthogonal polygons. In (general) simple polygons, there are exact polynomial-time algorithms; the current best running times are $O(n^3 \log n)$ for the *anchored* version (a starting point s which the route must pass through is given) and $O(n^4 \log n)$ for the *floating* version (no starting point is given) [5].

In some settings, complete coverage might not be feasible or necessary, thus we are also interested in computing a shortest route that sees at least an area of $A \geq 0$ within P. This is known as the Quota Watchman Route problem (QWRP), introduced in [8]. In contrast to the tractable WRP, the QWRP is (weakly) NP-hard, but a fully polynomial-time approximation scheme (FPTAS) is known. Any results about the QWRP can be adapted to the WRP by simply letting A be equal to the area of P.

We consider the generalization to multiple agents of both the WRP and the QWRP, namely the k-Watchmen Routes problem (k-WRP) and the Quota k-Watchmen Routes problem (Qk-WRP), with the objective of minimizing the length of the longest path traveled by any one watchman. Even in a simple polygon, when no starting points are specified (so, we are to determine the best starting locations), both problems are NP-hard to approximate within any multiplicative factor [12].

We thus focus on the (boundary) anchored version, in which a team of robots or searchers enter a domain P through a door on its boundary to search for a stationary target, which may be randomly distributed within the domain; the objective is to plan for an optimal collective effort to guarantee at least a certain probability of detection (1 in the k-WRP and some $p \in [0, 1]$ in the Qk-WRP). We consider the number, k, of robots to be fixed and relatively small, as in most practical situations it is infeasible to employ arbitrarily many

*Department of Applied Mathematics and Statistics, Stony Brook University, joseph.mitchell@stonybrook.edu

†Department of Applied Mathematics and Statistics, Stony Brook University, linh.nguyen.1@stonybrook.edu

watchmen/robots/agents. We present, for any fixed k, a pseudopolynomial-time (polynomial in the number n of vertices of P and the length of the longest edge of P) exact algorithm to solve the anchored k-WRP in a simple orthogonal (integral coordinate) polygon P under L1 distance. The pseudopolynomial-time exact algorithm is the basis for the FPTAS for L1 distance and the $(\sqrt{2}+\varepsilon)$-approximation for L2 distance. For the Qk-WRP, we give polynomial-time constant-factor approximations in a simple polygon. While we restrict ourselves to the anchored version, we achieve better approximation factors for any (fixed) k than the ones Nilsson and Packer proposed for the case $k = 2$ in [11].

2 Preliminaries

Let P be a *simple polygon*, i.e. a simply connected subset of $\mathbb{R}^2$. Denote by ∂P the boundary of P, a polygonal chain that does not self-intersect consisting of n vertices $v_1, v_2, \ldots, v_n$, which we assume to have integer coordinates. A simple polygon is *orthogonal* if the internal angle at every vertex is either 90 (convex vertex) or 270 degrees (reflex vertex).

For a point $x \in P$, its *visibility region*, denoted by $V(x)$, is the set of all points y such that the segment xy does not intersect with the exterior of P: we say x and y and see each other. For an arbitrary set $X \subseteq P$, the visibility region of X, $V(X)$, is the set of all points that are seen by some point in X. When X is either a point or a line segment, $V(X)$ is necessarily a simple subpolygon of P with at most n vertices and can be computed in $O(n)$ time [7, 13]. We use $|\cdot|$ to denote Euclidean measure of geometric objects (e.g., length or area).

The first problem we investigate is the anchored k-WRP, where the polygon P is orthogonal and movements of the watchmen are rectilinear (L1 distance). Given a simple orthogonal polygon P and a starting point $s \in \partial P$, we compute k tours $\{\gamma_i\}$ within P consisting of horizontal and vertical segments, all starting from s such that $\bigcup_{i=1,\ldots,k} V(\gamma_i) = P$ and $\max_{i=1,\ldots,k} |\gamma_i|$ is minimized. We also assume that the coordinates of the vertices of P are integers. It is known that even for $k = 2$, the general k-WRP in a simple polygon is (weakly) NP-hard via a simple reduction from Partition [10]. The reduction can be easily modified to show that our version is also NP-hard. The second problem, Qk-WRP, generalizes the first to $\left|\bigcup_{i=1,\ldots,k} V(\gamma_i)\right| \geq A$ for some $0 \leq A \leq |P|$. The fraction of area seen, $\frac{A}{|P|}$, can be interpreted as the probability that the watchmen detect a target uniformly distributed in P. We consider the Qk-WRP in a simple polygon, where the watchmen have unrestricted movement (not limited to horizontal and vertical).

3 k-Watchmen in a Simple Orthogonal Polygon

Dynamic programming exact algorithm A *visibility cut* c_i with respect to the starting point s is a chord obtained from extending the edge e incident on a reflex vertex, v_i, where e is the edge whose extension creates a convex vertex at v_i in the subpolygon containing s. The other subpolygon (not containing s) is the *pocket* induced by c_i. Not all reflex vertices induce a visibility cut. An *essential cut* is a visibility cut whose pocket does not fully contain any other pocket (Figure 1). In general, essential cuts may intersect with each other.

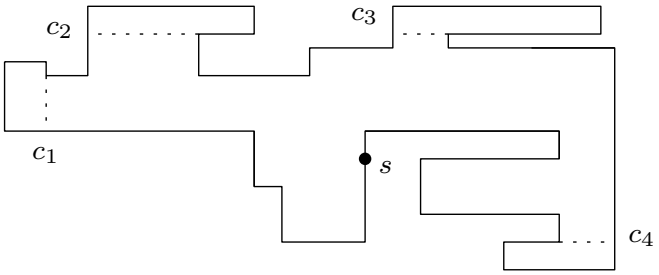


Figure 1: The essential cuts (dashed).

Lemma 1 $\bigcup_{i=1,\ldots,k} V(\gamma_i) = P$ *if and only if* $\{\gamma_i\}$ *collectively visit all essential cuts of* P.

Proof. The lemma is simply an extension of the well known fact: a single tour sees all of P if and only if it visits all essential cuts [2, 4, 5]. □

Denote by C_i the set of essential cuts visited by γ_i.

Corollary 2 *There exists an optimal solution* $\{\gamma_i\}$ *such that for any* i, γ_i *is the shortest route to visit all cuts in* C_i *and* s *in the order in which they appear around* ∂P.

Consider the decomposition of P into rectangular cells by the maximal (within P) extensions of all edges, as well as a horizontal and vertical line through s; this is known as the Hanan grid (Figure 2).

Lemma 3 *There exists an optimal solution* $\{\gamma_i\}$ *within the Hanan grid.*

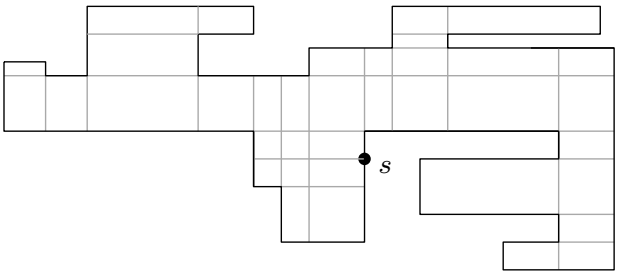


Figure 2: The Hanan grid formed by extensions of all edges in P.

Proof. Given an optimal solution $\{\gamma_i\}$, let $C_i = \{c_{i1}, \ldots, c_{ij}\}$ (in order around ∂P) and $p_{i1}, \ldots, p_{ij}$ be the point where γ_i first makes contact with $c_{i1}, \ldots, c_{ij}$. Denote by $L1_P(x, y)$ a geodesic L1 shortest path between x and y, a rectilinear shortest path constrained to stay within P. (For an overview of geodesic shortest paths in both L1 and L2 metrics, see [9].)

First, note that for every i, we may replace γ_i with a concatenation of geodesic L1 shortest paths, namely $\gamma_i := L1_p(s, p_{i1}) \cup L1_P(p_{i1}, p_{i2}) \cup \ldots \cup L1_P(p_{ij}, s)$ without increasing $\max_{i=1,\ldots,k} \{|\gamma_i|\}$ while maintaining visibility coverage of P.

We argue that $L1_p(s, p_{i1})$ is a geodesic L1 shortest path from s to c_{i1}. Suppose to the contrary, that geodesic L1 shortest paths from s to c_{i1} make contact with c_{i1} at $p'_{i1} \neq p_{i1}$ (all geodesic L1 shortest paths from a point to a segment have the same endpoint). Due to orthogonality $|L1_P(s, p'_{i1})| + |p'_{i1}p_{i1}| = |L1_P(s, p_{i1})|$, which means $|L1_P(s, p_{i1})| + |L1_P(p_{i1}, p_{i2})| = |L1_P(s, p'_{i1})| + |p'_{i1}p_{i1}| + |L1_P(p_{i1}, p_{i2})| \geq |L1_P(s, p'_{i1})| + |L1_P(p'_{i1}, p_{i2})|$. This implies γ_i should take a geodesic L1 shortest path from s to c_{i1}, and it suffices to find such a path within the Hanan grid. By a straightforward inductive argument, we can show the same for any portion of γ_i between any two essential cuts. □

Corollary 2 and Lemma 3 allow us to reduce the problem to that of finding a set of grid points on the essential cuts for which each route is responsible. Then, each route is simply the concatenation of L1 shortest paths between those points. Let $\{c_1, c_2, \ldots, c_m\}$ be the set of essential cuts in order around ∂P (s lies between c_1 and c_m). We define each subproblem $(c_j, p_1, l_1, \ldots, p_k, l_k)$ by an essential cut c_j, k Hanan grid points $p_1, \ldots, p_k$ on essential cuts $c_1, \ldots, c_j$ (and s) and k integers $l_1, \ldots, l_k$. Refer to Figure 3 for an illustration. Subproblem $(c_j, p_1, l_1, \ldots, p_k, l_k) =$ TRUE if and only if there exists a collection of k paths $\Gamma_1, \ldots, \Gamma_k$ collectively visiting all essential cuts from c_1 up to c_j such that

- Γ_i starts at s, ends at p_i,
- $|\Gamma_i| = l_i$.

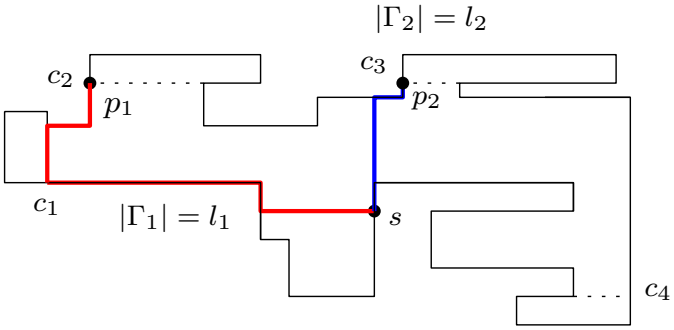


Figure 3: An example subproblem $(c_3, p_1, l_1, p_2, l_2)$.

The recursion is as follows. For each Hanan grid point $p \in c_j$ and $i = 1, \ldots, k$

$$(c_j, p_1, l_1, \ldots, p_i := p, l_i, \ldots, p_k, l_k) = \bigvee_{p'} (c_{j-1}, p_1, l_1, \ldots, p_i := p', l_i - |L1_P(p, p')|, \ldots, p_k, l_k) \quad (1)$$

where p' is taken from the set of all Hanan grid points on the cuts $c_1, \ldots, c_{j-1}$ such that geodesic L1 shortest paths from p' to c_j make contact with c_j at p (Lemma 3). The base case is simply $(s, s, 0, \ldots, s, 0) =$ TRUE. After tabulating all subproblems, we take the subproblem $(c_m, p_1, l_1, \ldots, p_k, l_k)$ (such that $(c_m, p_1, l_1, \ldots, p_k, l_k) =$ TRUE) with the minimum $\max_{i=1,\ldots,k} \{l_i + |L1_P(p_i, s)|\}$ and return the tours $\{\gamma_i := \Gamma_i \cup L1_P(p_i, s)\}$.

Proof of correctness Our proof of correctness relies on two arguments:

- Since the paths associated with subproblem $(c_{j-1}, p_1, l_1, \ldots, p_i := p', l_i - |L1_P(p, p')|, \ldots, p_k, l_k)$ visit all essential cuts up to c_{j-1}, the paths associated with subproblem $(c_j, p_1, l_1, \ldots, p_i := p, l_i, \ldots, p_k, l_k)$ also visit all essential cuts up to c_j since $p \in c_j$. By induction, the tours returned hence visit all essential cuts.
- γ_i consists of geodesic L1 shortest paths between contact points with essential cuts (proof of Lemma 3). If we identify two consecutive contact

points on γ_i, say p' and p in that order, then the length of the portion of γ_i from s to p is l_i if and only if the length of the portion of γ_i from s to p' is $l_i - |L1_P(p, p')|$.

Analysis of running time There are $O(n)$ essential cuts, $O(n)$ Hanan grid points on each cut. Each tour γ_i must be no longer than nD, where D is the length of the longest edge of P, therefore l_i is bounded by nD. In total, there are $O[n \cdot n^{2k} \cdot (nD)^k] = O(n^{3k+1}D^k)$ subproblems. We pre-compute geodesic L1 shortest paths between Hanan grid points, as well as between Hanan grid points and essential cuts, which equates to solving the All Pairs Shortest Path problem in the embedded graph of the Hanan grid. Then, we can solve each subproblem by iterating through at most $O(n^2)$ previously solved subproblems. Thus, the total running time is $O(n^{3k+3}D^k)$, which is pseudopolynomial for fixed k. This is in congruence with the weak NP-hardness from Partition, for which there exists a pseudopolynomial (polynomial in the number of input integers and the largest input integer) time algorithm. A tighter time bound is $O(n^{2k+3}L^k)$, where L is the length of a shortest single orthogonal watchman route of P, which is computable in $O(n)$ time if P is simple and orthogonal [3]. Clearly $L \leq |\partial P| \leq nD$ and $\max_{i=1,\ldots,k} |\gamma_i| \leq L$ (one shortest single watchman route and $k-1$ routes of length 0 is a feasible solution to the k-WRP). In addition, we need not consider any $L1_P(p, p')$ whose length is greater than L for recursion (1) of the dynamic programming.

Fully polynomial-time approximation scheme To achieve fully polynomial running time for fixed k, we bound the number of subproblems by "bucketing" the lengths of paths in P. Let $\{\gamma_i\}$ be an optimal collection of k routes. Consider that $L \leq \sum_{i=1,\ldots,k} |\gamma_i|$ (the concatenation of $\{\gamma_i\}$ can be considered a single watchman route) hence

$$\frac{L}{k} \leq \max_{i=1,\ldots,k} |\gamma_i| \leq L. \tag{2}$$

Given any $\varepsilon > 0$, we divide L into $\lceil \frac{nk}{\varepsilon} \rceil$ uniform intervals, each no longer than $\frac{\varepsilon L}{nk}$. The length of any geodesic L1 shortest path we take into consideration for recursion (1) must fall into one of the intervals, we round it down to the nearest interval endpoint. Then, apply the dynamic programming algorithm to the new instance with subproblems defined instead by intervals' endpoints. Let the solution returned be $\{\gamma_i'\}$. For clarity, we denote by $d(.)$ distance/length in the "rounded down" instance. Then

$$\max_{i=1,\ldots,k} |\gamma_i| \geq \max_{i=1,\ldots,k} d(\gamma_i) \geq \max_{i=1,\ldots,k} d(\gamma_i'). \tag{3}$$

The first inequality follows simply from the fact that we round down any distance from the original instance, the second inequality is by definition, since $\{\gamma_i'\}$ is an optimal solution of the new instance. Now, any route in $\{\gamma_i'\}$ must consist of at most n geodesic L1 shortest paths between Hanan grid points on essential cuts, the length of each differs by no more than $\frac{\varepsilon L}{nk}$ between the original instance and the "rounded down" instance. Thus, for any i

$$|\gamma_i'| - d(\gamma_i') \leq n \cdot \frac{\varepsilon L}{nk}$$

therefore

$$\max_{i=1,\ldots,k} d(\gamma_i') + \frac{\varepsilon L}{k} \geq \max_{i=1,\ldots,k} |\gamma_i'|. \tag{4}$$

Combining all three inequalities (2), (3), (4), we get

$$(1+\varepsilon) \max_{i=1,\ldots,k} |\gamma_i| \geq \max_{i=1,\ldots,k} |\gamma_i'|$$

with a running time of $O\left(n^{2k+3}\left(\frac{nk}{\varepsilon}\right)^k\right)$.

Remark The FPTAS for orthogonal movement (L1 distance) gives a polynomial time $(\sqrt{2}+\varepsilon)$-approximation to unrestricted movement (L2 distance).

Theorem 4 *For any fixed k, the anchored k-WRP in a simple orthogonal polygon has an FPTAS for the L1 metric and a polynomial-time $(\sqrt{2}+\varepsilon)$-approximation for the L2 metric.*

4 Quota k-Watchmen in a Simple Polygon

In this section, we assume P is a general simple polygon and the watchmen can move in any direction within P.

Constant factor polynomial-time approximation Let $\{\gamma_i\}$ be an optimal collection of quota k-watchman routes to achieve the visibility area quota of A and let $OPT = \max_{i=1,\ldots,k} |\gamma_i|$. Denote by $C_g(r)$ the geodesic disk of radius r centered at s, i.e. the locus of all points within geodesic distance (length of the geodesic shortest path) of r from s. Let $r = r_{\min}$, where $r_{\min}$ is the smallest value of r such that $|V(C_g(r))| \geq A$; $r_{\min}$

can be computed in $O(n^2 \log n)$ time using the "visibility wave" methods in [1]. Clearly, $r_{\min} \leq \frac{OPT}{2}$, since $C_g\left(\frac{OPT}{2}\right)$ encloses $\{\gamma_i\}$ and must see an area no smaller than A. If we repeatedly multiply r by 2, at some point we must have $\frac{r}{2} \leq \frac{OPT}{2} \leq r$, suppose we have reached this point. Then, $C_g(r)$ contains $\{\gamma_i\}$. Let γ be a shortest single route contained within $C_g(r)$ such that $|V(\gamma)| \geq A$ (note that γ is not necessarily the shortest single quota watchman route overall in P).

Lemma 5 $\frac{|\gamma|}{k} \leq OPT \leq |\gamma|$.

Proof. Recall that in Section 3, we proved two similar inequalities for watchman routes with orthogonal movement seeing the whole polygon. The same holds here since orthogonality and quota did not play a part in the argument. □

We show how to approximate γ (it is NP-hard to exactly compute γ), and that the number of times we multiply r by 2 is polynomial in n.

Lemma 6 *[8, Section 3] Given a budget $B \geq 0$ and any $\varepsilon > 0$, there exists an $O\left(\frac{n^5}{\varepsilon^6}\right)$ algorithm that computes a route of length at most $(1+\varepsilon)B$ seeing as much area as any route of length B within $C_g(r)$.*

We briefly describe the algorithm, and refer the readers to [8] for more details. First, triangulate P, including s as a vertex of the triangulation. Then, overlay onto the triangulation a regular square grid of side lengths $\delta = O\left(\frac{\varepsilon B}{n}\right)$ within an axis aligned square of size B-by-B centered at s. We consider the set of (convex) cells that overlap (both fully and partially) with $C_g(r)$ and their vertices, $S_{\delta,r}$. Let γ_B be the B-length route within $C_g(r)$ that achieves the most area of visibility. There exists a route of length at most $(1+\varepsilon)B$ with vertices coming from $S_{\delta,r}$ enclosing γ_B, i.e. the boundary of the relative convex hull (the minimum-perimeter connected superset within P, see [8, 9]) of the cells containing vertices of γ_B, thus seeing at least as much area as γ_B (Figure 4). If $|\gamma| \leq B \leq \alpha|\gamma|$ for some $\alpha \geq 1$, using dynamic programming, the algorithm in [8, Section 3] computes a route γ' of length no longer than $\alpha(1+\varepsilon)|\gamma|$ with vertices in $S_{\delta,r}$ that sees the most area, which must be no smaller than $|V(\gamma_B)| \geq |V(\gamma)| \geq A$. Using Lemma 7, we acquire a polynomial-sized set of values from which we can search for an appropriate B.

Lemma 7 $r_{\min} \leq |\gamma| \leq 6nr_{\min}$.

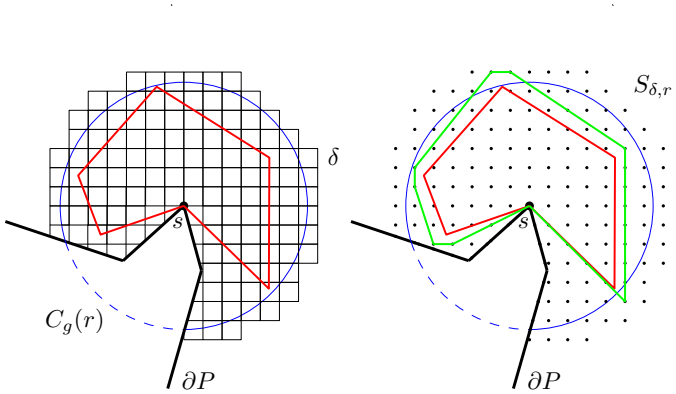


Figure 4: Left: γ_B (red) is a tour no longer than B within $C_g(r)$ (blue) that sees the most area. Right: enclosing γ_B with a tour whose vertices are in $S_{\delta,r}$ seeing everything γ_B sees (green).

Proof. The first inequality is straightforward, $r_{min} \leq OPT \leq |\gamma|$.

For the second inequality, first note that if a single watchman travels from s to $\partial C_g(r_{min})$, follows along the whole of $\partial C_g(r_{min})$ then goes back to s, he sees an area of A, thus $|\partial C_g(r_{min})| + 2r_{min}$ is an upper bound on $|\gamma|$. We show that $|\partial C_g(r_{min})| + 2r_{\min} \leq 6nr_{min}$. Observe that $\partial C_g(r_{min})$ consists of polygonal chains that are portions of ∂P and circular arcs; the circular arcs have total length no greater than $2\pi r_{min}$. Each segment in the polygonal part of $\partial C_g(r_{min})$ has length bounded by the sum of geodesic distances from its endpoints to s (triangle inequality), which is no more than $2r_{min}$. There are at most n segments in the polygonal portions of $\partial C_g(r_{min})$, therefore their total length is no greater than $2nr_{\min}$, implying $|\partial C_g(r_{min})| + 2r_{min} = 2nr_{min} + 2\pi r_{min} + 2r_{min} \leq 6nr_{min}$. □

We divide $6nr_{min}$ into $\lceil \frac{6n}{\varepsilon} \rceil$ uniform intervals so that each is no longer than εr_{min}: the smallest interval endpoint that is no smaller than $|\gamma|$ must also be no larger than $(1+\varepsilon)|\gamma|$, and hence is the value of B that we desire. We perform a binary search on the values $\left\{0, \frac{6nr_{min}}{\lceil \frac{6n}{\varepsilon} \rceil}, \ldots, 6nr_{min}\right\}$ as the input budget for the algorithm in Lemma 6, and seek out the smallest value for which the output route γ' sees an area no smaller than A. Clearly, $|\gamma'| \leq (1+\varepsilon)^2|\gamma|$.

Lemma 7 also implies that the number of times we double r is polynomially bounded, in particular, $O(\log n)$, since $r_{min} \leq OPT \leq 6nr_{min}$.

We are now ready to describe the approximation algorithm for the Qk-WRP as follows:

- Step 1: Set $r := r_{\min}$.
- Step 2: Compute γ', a $(1+\varepsilon)^2$-approximation to γ.
- Step 3: Divide γ' into k subpaths of equal length, each of which is bounded by $a_i, a_{i+1} \in \gamma'$ ($a_1 \equiv s \equiv a_{k+1}$) and denoted by $\gamma'_{a_i a_{i+1}}$.
- Step 4: For each i, we obtain γ'_i by traversing the geodesic shortest path from s to a_i, $\gamma'_{a_i a_{i+1}}$ and the geodesic shortest path from a_{i+1} back to s.
- Step 5: Set $r := 2r$, then repeat from Step 2, until $r > 6nr_{min}$.

Finally, we return the collection of routes $\{\gamma'_i\}$ that minimizes $\max_{i=1,\ldots,k} |\gamma'_i|$ out of all collections from all values of r in the doubling search.

Analysis of running time For each choice of B, we execute the $O\left(\frac{n^5}{\varepsilon^6}\right)$ algorithm, thus computing an approximation to γ' for each value of r takes $O\left(\frac{n^5}{\varepsilon^6}\log\left(\frac{n}{\varepsilon}\right)\right)$ time. This step dominates both computing $r_{\min}$ and deriving the collection $\{\gamma'_i\}$. Since there are $O(\log n)$ iterations of the doubling search for r, the overall running time is $O\left(\frac{n^5}{\varepsilon^6}\log\left(\frac{n}{\varepsilon}\right)\log n\right)$.

Theorem 8 *The algorithm described above has an approximation factor of* $3+\varepsilon$.

Proof. Since all our choices for B are no larger than $6nr_{\min}$, we can choose an appropriate $\delta = O\left(\frac{\varepsilon B}{n}\right)$ so that the geodesic distance from any point on γ' to s is no longer than $r+\varepsilon r$. Thus, when $\frac{r}{2} \le \frac{OPT}{2} \le r$, any one of the k routes returned by the algorithm is no longer than $\frac{|\gamma'|}{k} + 2r + 2\varepsilon r \le [(1+\varepsilon)^2 + 2 + 2\varepsilon]OPT = (3+\varepsilon')OPT$, where $\varepsilon' = 4\varepsilon + \varepsilon^2$. Note that $\frac{1}{\varepsilon} = \Theta\left(\frac{1}{\varepsilon'}\right)$ as ε and ε' approach 0, so the running time is in the same order when written in terms of ε'. □

Improving the approximation factor In the approximation algorithm earlier, we gradually expand $C_g(r)$ until $C_g(r)$ contains an optimal $\{\gamma_i\}$. If in each iteration, we instead multiply r by a smaller factor, namely $(1+\varepsilon)$, then at some point $\frac{r}{(1+\varepsilon)} \le \frac{OPT}{2} \le r$. The distance from each point on γ' to s is then no greater than $r + \varepsilon r \le (1+\varepsilon)^2\frac{OPT}{2}$. Hence, the length of any of the k routes returned by the approximation algorithm is bounded by $\frac{|\gamma'|}{k} + (1+\varepsilon)^2\frac{OPT}{2} + (1+\varepsilon)^2\frac{OPT}{2} \le (2+\varepsilon')OPT$, where $\varepsilon' = 4\varepsilon + 2\varepsilon^2$.

There is however, a trade-off between the approximation factor and the number of iterations of the multiplicative search for r. If we multiply r by $(1+\varepsilon)$ each time, the search requires $O(\log_{1+\varepsilon} n)$ iterations. Note that

$$\log_{1+\varepsilon} n = \log n \frac{\ln 2}{\ln(1+\varepsilon)} = \log n O\left(\frac{1}{\varepsilon}\right).$$

In summary, we can achieve an approximation ratio of $(2+\varepsilon')$ with a running time of $O\left(\frac{n^5}{\varepsilon^7}\log\left(\frac{n}{\varepsilon}\right)\log n\right) = O\left(\frac{n^5}{\varepsilon'^7}\log\left(\frac{n}{\varepsilon'}\right)\log n\right)$ (since $\frac{1}{\varepsilon} = \Theta\left(\frac{1}{\varepsilon'}\right)$).

References

[1] Esther M. Arkin, Alon Efrat, Christian Knauer, Joseph S. B. Mitchell, Valentin Polishchuk, Günter Rote, Lena Schlipf, and Topi Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7(2):77–100, 2016.

[2] Svante Carlsson, Håkan Jonsson, and Bengt J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete & Computational Geometry*, 22:377–402, 1999.

[3] Wei-Pang Chin and Simeon Ntafos. Optimum watchman routes. In *Proceedings of the 2nd Annual Symposium on Computational Geometry*, pages 24–33, 1986.

[4] Wei-Pang Chin and Simeon Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6(1):9–31, 1991.

[5] Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 473–482, 2003.

[6] Adrian Dumitrescu and Csaba D. Tóth. Watchman tours for polygons with holes. *Computational Geometry*, 45(7):326–333, 2012.

[7] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *Proceedings of the 2nd Annual Symposium on Computational Geometry*, pages 1–13, 1986.

[8] Kien C. Huynh, Joseph S. B. Mitchell, Linh Nguyen, and Valentin Polishchuk. Optimizing visibility-based search in polygonal domains. In *Proceedings of the 19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:16, 2024.

[9] Joseph S. B. Mitchell. Geometric shortest paths and network optimization. *Handbook of Computational Geometry*, 334:633–702, 2000.

[10] Joseph S. B. Mitchell and Erik L. Wynters. Watchman routes for multiple guards. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, pages 126–129, 1991.

[11] Bengt J. Nilsson and Eli Packer. Approximation algorithms for the two-watchman route in a simple polygon. *arXiv preprint arXiv:2309.13428*, 2023.

[12] Eli Packer. Computing multiple watchman routes. In *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7*, pages 114–128. Springer, 2008.

[13] Csaba D. Tóth, Joseph O'Rourke, and Jacob E. Goodman. *Handbook of Discrete and Computational Geometry*. CRC press, 2017.

An Improved Algorithm for Shortest Paths in Weighted Unit-Disk Graphs*

Bruce W. Brewer† Haitao Wang‡

Abstract

Let V be a set of n points in the plane. The unit-disk graph $G = (V, E)$ has vertex set V and an edge $e_{uv} \in E$ between vertices $u, v \in V$ if the Euclidean distance between u and v is at most 1. The weight of each edge e_{uv} is the Euclidean distance between u and v. Given V and a source point $s \in V$, we consider the problem of computing shortest paths in G from s to all other vertices. The previously best algorithm for this problem runs in $O(n \log^2 n)$ time [Wang and Xue, SoCG'19]. The problem has an $\Omega(n \log n)$ lower bound under the algebraic decision tree model. In this paper, we present an improved algorithm of $O(n \log^2 n / \log\log n)$ time (under the standard real RAM model). Furthermore, we show that the problem can be solved using $O(n \log n)$ comparisons under the algebraic decision tree model, matching the $\Omega(n \log n)$ lower bound.

1 Introduction

Let V be a set of n points in the plane. The unit-disk graph $G = (V, E)$ has vertex set V and an edge $e_{uv} \in E$ between vertices $u, v \in V$ if the Euclidean distance between u and v is at most 1. Alternatively, G can be seen as the intersection graph of disks with radius $\frac{1}{2}$ centered at the points in V (i.e., two disks have an edge in the graph if they intersect). In the *weighted graph*, the weight of each edge $e_{uv} \in E$ is the Euclidean distance between u and v. In the *unweighted graph*, all edges have the same weight.

Given V and a source point $s \in V$, we study the single source shortest path (SSSP) problem where the goal is to compute shortest paths from s to all other vertices in G. Like in general graphs, the algorithm usually returns a shortest path tree rooted at s. The problem in the unweighted graph has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model since even deciding if G is connected requires that much time by a reduction from the max-gap [2]. The unweighted problem has been solved optimally in $O(n \log n)$ time by Cabello and Jejčič [2], or in $O(n)$ time by Chan and Skrepetos [4] if the points of V are pre-sorted (by both the x- and y-coordinates). Several algorithms for the weighted case are also known [2, 8, 11, 13, 15]. Roditty and Segal [13] first solved the problem in $(n^{4/3+\delta})$ time, where $\delta > 0$ is an arbitrarily small constant. Cabello and Jejčič [2] improved it to $O(n^{1+\delta})$ time. Subsequent improvements were made by Kaplan, Mulzer, Roditty, Seiferth, and Sharir [8] and also by Liu [11] by developing more efficient dynamic bichromatic closest pair data structures and plugging them into the algorithm of [2]. Wang and Xue [15] proposed a new method that solves the problem in $O(n \log^2 n)$ time without using dynamic bichromatic closest pair data structures. It is currently the best algorithm for the problem.

1.1 Our result

We present a new algorithm of $O(n \log^2 n / \log\log n)$ time for the weighted case and, therefore, slightly improve the result of [15]. Our algorithm follows the framework of Wang and Xue [15] but provides a more efficient solution to a bottleneck subproblem in their algorithm, called the *offline insertion-only additively-weighted nearest neighbor problem with a separating line* (or IOAWNN-SL for short). Specifically, we are given a sequence of n operations of the following two types: (1) Insertion: Insert a weighted point to P (which is $\emptyset$ initially); (2) Query: given a query point q, find the *additively-weighted nearest neighbor* of q in P, where the distance between q to any point $p \in P$ is defined to be their Euclidean distance plus the weight of p. The points of P and all the query points are required to be separated by a given line (say the x-axis). The goal of the problem is to answer all queries.

Wang and Xue [15] solved the IOAWNN-SL problem in $O(n \log^2 n)$ time using the traditional logarithmic method of Bentley [1]. This is the bottleneck of their overall shortest path algorithm; all other parts of the algorithm take $O(n \log n)$ time. We derive a more efficient algorithm that solves IOAWNN-SL in $O(n \log^2 n / \log\log n)$ time (see Theorem 1 for details). Plugging this result into the algorithm framework of Wang and Xue [15] solves the shortest path problem in $O(n \log^2 n / \log\log n)$ time.

Theorem 1 *Let P be an initially empty set of n weighted points in the plane such that all points of P lie below the x-axis ℓ. There exists a data structure $\mathcal{D}(P)$*

*This research was supported in part by NSF under Grant CCF-2005323.

†Kahlert School of Computing, University of Utah, Salt Lake City, UT 84112, USA. bruce.brewer@utah.edu

‡Kahlert School of Computing, University of Utah, Salt Lake City, UT 84112, USA. haitao.wang@utah.edu

of $O(n)$ space supporting the following operations:

1. *Insertion: Insert a weighted point p below ℓ to P in amortized $O(\log^2 n/\log\log n)$ time.*
2. *Query: Given a query point q above ℓ, find the additively-weighted nearest neighbor to q in P in worst-case $O(\log^2 n/\log\log n)$ time.*

Our algorithm for Theorem 1 needs to solve a subproblem about merging two additively weighted Voronoi diagrams. Specifically, let S_a and S_b each be a subset of n weighted points in the plane such that all points of $S_a \cup S_b$ are below the x-axis ℓ. Let $\mathcal{VD}(S_a)$ denote the additively-weighted Voronoi diagram of S_a, and $\mathcal{VD}_+(S_a)$ denote the portion of $\mathcal{VD}(S_a)$ above ℓ. Similarly, define $\mathcal{VD}(S_b)$ and $\mathcal{VD}_+(S_b)$ for S_b, and define $\mathcal{VD}(S_a \cup S_b)$ and $\mathcal{VD}_+(S_a \cup S_b)$ for $S_a \cup S_b$. Given $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$, our problem is to compute $\mathcal{VD}_+(S_a \cup S_b)$. We solve the problem in $O(n)$ time by modifying Kirkpatrick's algorithm for merging two standard Voronoi diagrams [9] and by making use of the property that $\mathcal{VD}_+(S_a \cup S_b)$ and all points of $S_a \cup S_b$ are separated by ℓ. Note that directly applying Kirkpatrick's algorithm does not work (see Section 3 for more details). It would be more interesting to have a linear time algorithm to compute the complete diagram $\mathcal{VD}(S_a \cup S_b)$ by merging $\mathcal{VD}(S_a)$ and $\mathcal{VD}(S_b)$. Our technique, however, does not immediately work because it relies on the separating line ℓ. Nevertheless, we hope our result will serve as a stepping stone towards achieving that goal. We summarize our result in the following theorem.

Theorem 2 *Let S_a and S_b each be a set of n weighted points in the plane such that all the points of $S_a \cup S_b$ are below the x-axis ℓ. Given $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$, $\mathcal{VD}_+(S_a \cup S_b)$ can be constructed in $O(n)$ time.*

Algebraic decision tree model. The above result holds for the standard real RAM model. Under the algebraic decision tree model in which we only count comparisons toward the time complexity, using a technique recently developed by Chan and Zheng [5], we show that the problem IOAWNN-SL can be solved using $O(n \log n)$ comparisons. This leads to an $O(n \log n)$ time algorithm for the shortest path problem in weighted unit-disk graphs under the algebraic decision tree model, matching the $\Omega(n \log n)$ lower bound [2].

Outline. The rest of the paper is organized as follows. We describe the shortest path algorithm framework in Section 2, mainly by reviewing Wang and Xue's algorithm [15]. In Section 3, we introduce our data structure for IOAWNN-SL and thus prove Theorem 1. Section 4 presents our Voronoi diagram merging algorithm for Theorem 2. We describe the algebraic decision tree algorithm in Section 5.

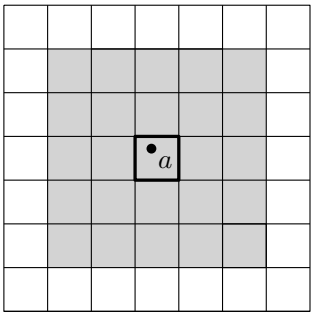


Figure 1: Illustrating $\square_a$ (the central highlighted square) and $\boxplus_a$ (the gray area).

2 The shortest path algorithm

In this section, we describe the shortest path algorithm. We begin with reviewing Wang and Xue's algorithm [15] and explain why the IOAWNN-SL problem is a bottleneck (we only state their algorithm and refer the interested reader to their paper [15] for the correctness analysis). We will show how our solution to IOAWNN-SL in Theorem 1 can lead to an $O(n \log^2 n/\log\log n)$ time algorithm for the shortest path problem.

Given a set V of n points in $\mathbb{R}^2$ and a source point $s \in V$, we wish to compute shortest paths from s to all vertices in the weighted unit-disk graph $G = (V, E)$. We use $e_{uv} \in E$ to denote the edge between two points $u, v \in V$ and $w(e_{uv})$ to denote the weight of the edge. Recall that $w(e_{uv}) = ||u-v|| \leq 1$, where $||u-v||$ denotes the Euclidean distance between u and v. The algorithm will compute a table dist$[\cdot]$ such that after the algorithm finishes, dist$[v]$ is the length of a shortest path from s to v for all $v \in V$. Using a predecessor table, we could also maintain a shortest path tree, but we will omit the discussion about it.

We overlay the plane with a grid Γ of square cells with side lengths $1/2$. For any point $a \in \mathbb{R}^2$, denote by $\square_a$ the cell of Γ such that $a \in \square_a$, and $\boxplus_a$ the 5×5 patch of cells in Γ centered around $\square_a$ (see Figure 1). For a set of points $A \subseteq \mathbb{R}^2$ and $a \in A$, we use $A_{\square_a} = A \cap \square_a$ and $A_{\boxplus_a} = A \cap \boxplus_a$. The algorithm makes use of the following properties: (1) For any two points a, b in the same cell of Γ, $||a - b|| \leq 1$ holds; (2) if $||a - b|| \leq 1$, then b is in $\boxplus_a$ and a is in $\boxplus_b$.

Wang and Xue's algorithm is summarized in Algorithm 1. It can be understood by contrasting with Dijkstra's algorithm, which we write in Algorithm 2 using similar notation. In particular, a subroutine UPDATE(A, B) is used to "push" the current candidate shortest path information from A to B where $A, B \subseteq V$. Specifically, for each point $b \in B$, we find:

$$p_b = \operatorname*{arg\,min}_{\{a \in A : e_{ab} \in E\}} \text{dist}[a] + w(e_{ab}). \tag{1}$$

We then update dist$[b]$ to $\min\{\text{dist}[b], \text{dist}[p_b] + w(e_{p_b b})\}$.

Algorithm 1: Wang and Xue's algorithm [15]

```
1 dist[a] ← ∞ for all a ∈ V
2 dist[s] ← 0
3 A ← V
4 while A ≠ ∅ do
5 |   c ← arg min_{a∈A}{dist[a]}
6 |   UPDATE(A_{⊞_c}, A_{□_c})        // First Update
7 |   UPDATE(A_{□_c}, A_{⊞_c})        // Second Update
8 |   A ← A \ A_{□_c}
9 return dist[·]
```

Algorithm 2: Dijkstra's algorithm

```
1 dist[a] ← ∞ for all a ∈ V
2 dist[s] ← 0
3 A ← V
4 while A ≠ ∅ do
5 |   c ← arg min_{a∈A}{dist[a]}
6 |   UPDATE({c}, A)
7 |   A ← A \ {c}
8 return dist[·]
```

The main difference between Wang and Xue's algorithm and Dijkstra's is that instead of operating on single vertices, Wang and Xue's algorithm operates on cells of Γ. Generally speaking, the first update (Line 6) in Algorithm 1 is to update the shortest path information for the points in $A_{\square_c}$ using the shortest path information of their neighbors. The second update is to use the shortest path information for the points in $V_{\square_c}$ to update the shortest path information of their neighbors. Wang and Xue prove that after the first update, the shortest path information for all points of $V_{\square_c}$ is correctly computed.

Wang and Xue give an $O(n \log^2 n)$ time solution for the second update, i.e., Line 7. The rest of Algorithm 1 takes $O(n \log n)$ time. We will improve the runtime for the second update to $O(n \log^2 n / \log\log n)$ using Theorem 1, which improves the runtime for Algorithm 1 to $O(n \log^2 n / \log\log n)$. The details are discussed in the following.

2.1 The second update

To implement the second update $\textsc{Update}(A_{\square_c}, A_{\boxplus_c})$, since $A_{\boxplus_c}$ has $O(1)$ cells, it suffices to perform $\textsc{Update}(A_{\square_c}, A_{\square})$ for each cell $\square \in \boxplus_c$ individually.

If $\square$ is $\square_c$, then $A_{\square_c} = A_{\square}$. Since the distance between two points in $\square_c$ is at most 1, $\textsc{Update}(A_{\square_c}, A_{\square})$ can be performed in $O(|A_{\square_c}| \log |A_{\square_c}|)$ time (and $O(|A_{\square_c}|)$ space) by constructing the additively-weighted Voronoi diagram for $A_{\square_c}$ [7].

If $\square$ is not $\square_c$, a useful property is that $\square$ and $\square_c$ are separated by an axis-parallel line. To perform $\textsc{Update}(A_{\square_c}, A_{\square})$, Wang and Xue [15] proposed Algorithm 3 below.

Algorithm 3: $\textsc{Update}(A, B)$ from [15]

```
1 dist'[a] ← dist[a] for a ∈ A
2 Sort the points in A = {a_1, ..., a_{|A|}} so that
    dist'[a_1] ≤ ... ≤ dist'[a_{|A|}]
3 for i = 1, ..., |A| do
4 |   B_i ← {b ∈ B : e_{a_i b} ∈ E and e_{a_j b} ∉
  |     E for all j < i}
5 U ← ∅
6 for i = |A|, ..., 1 do
7 |   U ← U ∪ {a_i}
8 |   for b ∈ B_i do
9 |   |   p = arg min_{u∈U}{dist'[u] + w(e_{ub})}
10|   |   dist[b] ← min{dist[b], dist'[p] + w(e_{pb})}
```

The correctness of Algorithm 3 hinges on the fact that p found by Line 9 is the same as p_b in Equation (1). This is seen by arguing that $p_b \in U$ and $e_{pb} \in E$.

We now analyze the runtime of Algorithm 3. Sorting A takes $O(|A| \log |A|)$ time. Computing the subsets B_i, $1 \le i \le |A|$, can be done in $O((|A|+|B|)\log(|A|+|B|))$ time (and $O(|A|+|B|)$ space) [15]. The for loop (Lines 6–10) is an instance of the IOAWNN-SL problem introduced in Section 1. Indeed, if we assign each point u in U a weight equal to $\text{dist}'[u]$, then p in Line 9 is essentially the additively-weighted nearest neighbor of b in U. The set U is dynamically changed with point insertions in Line 7. As such, by Theorem 1, the for loop can be implemented in $O(k \log^2 k / \log\log k)$ time (and $O(k)$ space) with $k = |A|+|B|$. Therefore, $\textsc{Update}(A_{\square_c}, A_{\square})$ can be performed in $O(k \log^2 k / \log\log k)$ time and $O(k)$ space, with $k = |A_{\square_c}| + |A_{\square}|$.

In summary, since $A_{\boxplus_c}$ has $O(1)$ cells, the second update $\textsc{Update}(A_{\square_c}, A_{\boxplus_c})$ can be implemented in $O(|A_{\boxplus_c}| \log^2 |A_{\boxplus_c}| / \log\log |A_{\boxplus_c}|)$ time and $O(|A_{\boxplus_c}|)$ space as $A_{\square_c} \subseteq A_{\boxplus_c}$. As analyzed in [15], the total sum of $|A_{\boxplus_c}|$ in the entire Algorithm 1 is $O(n)$. This leads to the following result.

Theorem 3 *Given a set V of n points in the plane and a source point s, shortest paths from s to all other vertices in the weighted unit-disk graph $G = (V, E)$ can be computed in $O(n \log^2 n / \log\log n)$ time and $O(n)$ space.*

3 The offline insertion-only additively-weighted nearest neighbor problem with a separating line (IOAWNN-SL)

In this section, we prove Theorem 1. We follow the notation in Section 1. In particular, for any subset $P' \subseteq P$, $\mathcal{VD}_+(P')$ denotes the portion of the additively-weighted Voronoi diagram of P' above the x-axis ℓ.

Our data structure $\mathcal{D}(P)$ for Theorem 1 consists of two components: $\mathcal{D}(P')$ and $\mathcal{VD}_+(P \setminus P')$ for some

subset $P' \subseteq P$; we maintain the invariant $|P'| \leq |P|/\log|P|$. We also build a point location data structure on $\mathcal{VD}_+(P \setminus P')$ so that, given a query point, the cell of $\mathcal{VD}_+(P \setminus P')$ containing the point can be found in $O(\log|P \setminus P'|)$ time [6, 10]. As such, $\mathcal{D}(P)$ is a recursive structure: $\mathcal{D}(P)$ is defined in terms of $\mathcal{D}(P')$ which in turn is defined in terms of $\mathcal{D}(P'')$ and so on. As the base case, if $|P| \leq c$ for some constant c, then we simply let $\mathcal{D}(P) = \mathcal{VD}_+(P)$. Similar recursive data structures have been used before in the literature, e.g., [3, 12].

In the following, we discuss how to handle the two operations: insertions and queries.

Queries. Given a query point q above ℓ, we first find the nearest neighbor of q in $P \setminus P'$ using a point location query on $\mathcal{VD}_+(P \setminus P')$. Then, we find the nearest neighbor of q in P' using $\mathcal{D}(P')$ recursively. Among the two "candidate" neighbors, we return the one nearer to q as the answer. For the query time, since a point location query on $\mathcal{VD}_+(P \setminus P')$ takes $O(\log|P \setminus P'|)$ time, the query time $Q(n)$ satisfies the following recurrence: $Q(n) = Q(n/\log n) + O(\log n)$, which solves to $Q(n) = O(\log^2 n/\log\log n)$. Therefore, each query operation takes worst-case $O(\log^2 n/\log\log n)$ time.

Insertions. To insert a point p below ℓ to P, we first insert p to P' recursively. We then check if the invariant $|P'| \leq |P|/\log|P|$ still holds. If not, we set $P' = \emptyset$, and then construct $\mathcal{VD}_+(P)$ as follows. First, we construct $\mathcal{VD}_+(P')$ recursively. Recall that $\mathcal{VD}_+(P \setminus P')$ is already available. We compute $\mathcal{VD}_+(P)$ by merging $\mathcal{VD}_+(P')$ and $\mathcal{VD}_+(P \setminus P')$, which takes $O(|P|)$ time by Theorem 2. Finally, we construct a point location data structure on $\mathcal{VD}_+(P)$ in $O(|P|)$ time [6, 10]. This finishes the insertion operation.

We now analyze the insertion time. First, suppose that we need to construct $\mathcal{VD}_+(P)$ due to the insertion of p. Then, the construction time $T(n)$ for $\mathcal{VD}_+(P)$ satisfies the following recurrence: $T(n) = T(n/\log n) + O(n)$, which solves to $T(n) = O(n)$.

Since $P' = \emptyset$ once $\mathcal{VD}_+(P)$ is constructed, we only need to construct $\mathcal{VD}_+(P)$ every $\Theta(n/\log n)$ insertions. As constructing $\mathcal{VD}_+(P)$ takes $O(|P|)$ time, the amortized time for constructing $\mathcal{VD}_+(P)$ per insertion is $O(\log n)$. As such, if $I(n)$ is the amortized time for each insertion, we have the following recurrence: $I(n) = I(n/\log n) + O(\log n)$. The recurrence solves to $I(n) = O(\log^2 n/\log\log n)$. We conclude that each insertion takes $O(\log^2 n/\log\log n)$ amortized time.

Note that the space $S(n)$ of $\mathcal{D}(P)$ satisfies the following recurrence: $S(n) = S(n/\log n) + O(n)$, which solves to $S(n) = O(n)$. This proves Theorem 1.

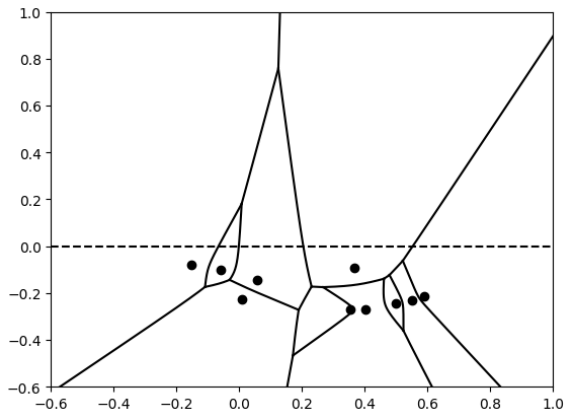


Figure 2: Illustrating an additively-weighted Voronoi diagram. The dashed horizontal line is the x-axis ℓ.

4 Merging two additively-weighted Voronoi diagrams

In this section, we prove Theorem 2. For completeness, we first introduce the formal definition of additively-weighted Voronoi diagrams and then present our merging algorithm.

4.1 Additively-weighted Voronoi diagrams

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n points in the plane such that each point s_i has a weight w_i that can be positive, zero, or negative. Following the literature, we refer to points of S as *sites*. We define the additively-weighted Euclidean distance (or *weighted distance* for short) of a point $p \in \mathbb{R}^2$ to a site s_i as $d(s_i, p) = ||s_i - p|| + w_i$.

The additively-weighted Voronoi diagram of S, denoted by $\mathcal{VD}(S)$, partitions the plane into Voronoi regions, Voronoi edges, and Voronoi vertices; see Figure 2. Each Voronoi region R_i is associated with a site s_i and is defined to be the set of points that are closer to s_i than to any other site measured by the weighted distances:

$$R_i = \{p \in \mathbb{R}^2 : d(s_i, p) < d(s_j, p), \forall j \neq i\}.$$

Each Voronoi edge E_{ij} is associated with two distinct sites s_i and s_j and is defined to be the set of points that are equidistant to s_i and s_j and closer to these sites than any other sites:

$$E_{ij} = \{p \in \mathbb{R}^2 : d(s_i, p) = d(s_j, p) < d(s_k, p), \forall k \neq i, j\}.$$

Each Voronoi vertex is associated with three or more distinct sites and is defined to be the point that is equidistant to these sites and closer to these sites than any other site.

We will also talk about the *bisector* between two sites, which is defined to be the set of points in the plane that

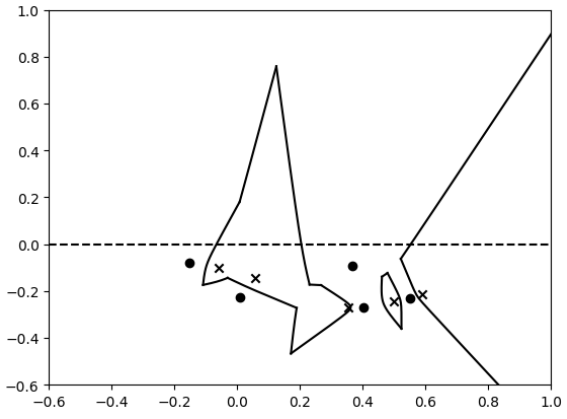


Figure 3: Illustrating the contour between two sets of points. The dashed horizontal line is the x-axis ℓ.

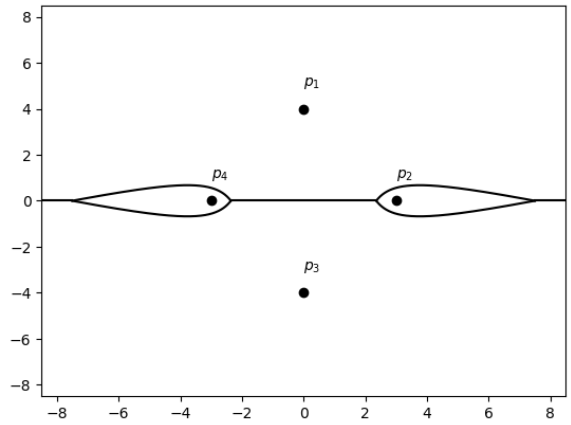


Figure 4: Illustrating the additively-weighted Voronoi diagram of four points $\{p_1, p_2, p_3, p_4\}$ for Observation 2.

are equidistant to the two sites:

$$B(s_i, s_j) = \{p \in \mathbb{R}^2 : d(s_i, p) = d(s_j, p)\}.$$

$B(s_i, s_j)$ is a hyperbolic arc whose foci are s_i and s_j. Note that a Voronoi edge associated with two sites is a subset of their bisector.

Observation 1 states some properties about $\mathcal{VD}(S)$ that are well known in the literature; we will use these properties in our algorithm.

Observation 1 ([7])

1. *Every Voronoi region of $\mathcal{VD}(S)$ must contain its associated site.*
2. *Each Voronoi region R_i of $\mathcal{VD}(S)$ is star-shaped with respect to its site s_i, that is, the line segment $\overline{s_i p}$ is inside R_i for any point $p \in R_i$.*
3. *The combinatorial size of $\mathcal{VD}(S)$ is $O(|S|)$.*

4.2 Merging algorithm for Theorem 2

We follow the notation introduced in Section 1, e.g., ℓ, n, S_a, S_b, $\mathcal{VD}(S_a)$, $\mathcal{VD}(S_b)$, $\mathcal{VD}_+(S_a)$, $\mathcal{VD}_+(S_b)$, etc. Let $S = S_a \cup S_b$. Given $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$, our goal is to compute $\mathcal{VD}_+(S)$ in $O(n)$ time. For ease of exposition, we make a general position assumption that no point in the plane is equidistant to four points of S.

Our strategy is to identify the *contour* which consists of edges in the complete Voronoi diagram $\mathcal{VD}(S)$ that are associated with a site in S_a and a site in S_b; see Figure 3. Note that the contour may have multiple connected components. The contour partitions the plane into regions CR_i such that $\mathcal{VD}(S) \cap CR_i$ is either $\mathcal{VD}(S_a) \cap CR_i$ or $\mathcal{VD}(S_b) \cap CR_i$ (we show in Lemma 4 later that each contour component has the topology of a line or a circle). As such, once we have identified the contour, computing $\mathcal{VD}(S)$ is straightforward. To compute the contour, the idea is to first find a point on each contour component and then trace the component by traversing $\mathcal{VD}(S_a)$ and $\mathcal{VD}(S_b)$ simultaneously. This strategy follows Kirkpatrick's algorithm [9] for merging two standard Voronoi diagrams. However, we cannot directly apply Kirkpatrick's algorithm because his method for finding a point in each contour component is not applicable to the weighted case. More specifically, his method relies on the property that the Euclidean minimum spanning tree of a point set in the plane must be a subgraph of the dual graph of its standard Voronoi diagram. However, this is not true anymore for the additively-weighted Voronoi diagrams. We make it formally as an observation below.

Observation 2 *The Euclidean minimum spanning tree of a set of points in the plane is not necessarily a subgraph of the dual graph of the additively-weighted Voronoi diagram of the point set.*

Proof. Figure 4 gives an example for the observation with $S = \{p_1, p_2, p_3, p_4\}$. It is obtained by setting $p_1 = (0, 4)$, $p_2 = (3, 0)$, $p_3 = (0, -4)$, and $p_4 = (-3, 0)$ with weights $w_1 = -4$, $w_2 = 0$, $w_3 = -4$, and $w_4 = 0$. Since (p_2, p_4) is the closest pair among the four points of S, $\overline{p_2 p_4}$ must be an edge in the Euclidean minimum spanning tree of S. However, there is no edge between p_2 and p_4 in the dual graph of the additively-weighted Voronoi diagram of S because their Voronoi regions are not adjacent. □

In our problem, we are interested in merging $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$ into $\mathcal{VD}_+(S)$, so it suffices to compute the portions of the contour above ℓ. With the help of ℓ, it is relatively easy to find a point on each contour component using the following property proved in Lemma 5: Every contour component above ℓ must intersect ℓ.

At a high level, our algorithm has two main procedures. The first one is to identify the intersections between the contour and ℓ. The second procedure is to start at these intersection points and trace each component of the contour above ℓ.

The first main procedure: Finding intersections between the contour and ℓ. By definition, ℓ is divided into segments by its intersections with $\mathcal{VD}_+(S_a)$, which we call ℓ-*edges* of $\mathcal{VD}_+(S_a)$; similarly, we define ℓ-*edges* for $\mathcal{VD}_+(S_b)$. We sweep ℓ from left to right, looking for places where the contour intersects ℓ. We start with the leftmost ℓ-edge of $\mathcal{VD}_+(S_a)$ and the leftmost ℓ-edge of $\mathcal{VD}_+(S_b)$. At each step, we are on some ℓ-edge e_a of $\mathcal{VD}_+(S_a)$ and some ℓ-edge e_b of $\mathcal{VD}_+(S_b)$. Let $s_a \in S_a$ be the site associated with the cell of $\mathcal{VD}_+(S_a)$ containing e_a; define $s_b \in S_b$ similarly. We compute the bisector $B(s_a, s_b)$ and determine where it intersects ℓ. The bisector is a hyperbolic arc and ℓ is a straight line, so they have at most two intersections p_1 and p_2. If $p_i \in e_a \cap e_b$, then p_i is a point of intersection between the contour and ℓ. In this way, we can compute all intersections between ℓ and the contour. Since the combinatorial sizes of $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$ are $O(n)$, this procedure computes $O(n)$ intersections between ℓ and the contour in $O(n)$ time.

The second main procedure: Tracing the contour. We trace the contour components from the intersection points computed above. Specifically, for each intersection p, we trace the contour component containing p as follows. Suppose that p is on an ℓ-edge e_a of $\mathcal{VD}_+(S_a)$ and an ℓ-edge e_b of ℓ in $\mathcal{VD}_+(S_b)$. These edges are associated with sites $s_a \in S_a$ and $s_b \in S_b$. Our trace begins at p and continues above ℓ along the bisector $B(s_a, s_b)$. This bisector enters a Voronoi region R_a of $\mathcal{VD}_+(S_a)$ and a region R_b of $\mathcal{VD}_+(S_b)$. We find which edge of R_a or R_b the bisector intersects first. If no intersection exists or the bisector first intersects ℓ, then we finish the trace by reporting that the portion of $B(s_a, s_b)$ past p is an edge of the contour. Otherwise, assume that we intersect an edge e'_a of R_a before an edge of R_b (the case where we intersect an edge of R_b first is handled the same way) and denote this point of intersection by p'. We rule out the case where $B(s_a, s_b)$ intersects a vertex instead of an edge because if we were to intersect a vertex, this vertex would be equidistant to three sites in S_a and one site in S_b, which would contradict our general position assumption that no point is equidistant to four sites of $S = S_a \cup S_b$. We report that the portion of $B(s_a, s_b)$ between p and p' is an edge of the contour. Then, we rename R_a to be the Voronoi region of $\mathcal{VD}_+(S_a)$ on the other side of e'_a and update $p \leftarrow p'$. We then continue the tracing from p following the same process as above.

Our tracing algorithm is similar to the well-known algorithm for merging the standard Voronoi diagrams of two sets of points separated by a line [14]. One difficulty with our algorithm is efficiently determining which edges of R_a and R_b the contour intersects first. This may not be a constant time operation since R_a and R_b may have many edges. The merge algorithm by Shamos and Hoey [14] takes advantage of the fact that the contour in their problem is monotone so that they can find all contour edges in a region by a single scan of the boundary of that region. In our problem, the contour may not be monotone. To resolve the issue, we follow the same technique used by Kirkpatrick [9] for merging standard Voronoi diagrams of two arbitrary sets of points. Specifically, before our tracing algorithm, we subdivide Voronoi regions of $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$ each into sub-regions of at most four edges by drawing segments between each site and each vertex of the Voronoi region of the site (see Figure 5; we can do this because each Voronoi region is star-shaped by Observation 1); as in [9], we refer to these segments as *spokes*. Because each sub-region only has at most four edges, finding where a bisector intersects a sub-region can be done in $O(1)$ time. We then apply our above tracing algorithm using these subdivisions of $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$. Each tracing step now finds an intersection between the contour and either a spoke or a Voronoi edge in constant time. As such, the total time of the tracing procedure is linear in the number of such intersections. By Lemma 6, the number of such intersections, and hence the runtime of the tracing procedure, is $O(n)$. Therefore, the total time of the algorithm for merging $\mathcal{VD}_+(S_a)$ and $\mathcal{VD}_+(S_b)$ is $O(n)$. This proves Theorem 2.

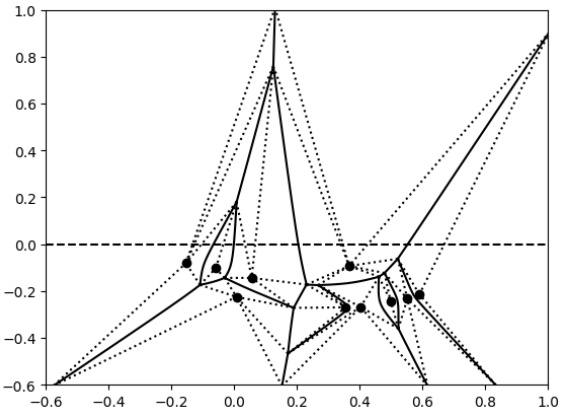


Figure 5: The dotted segments are spokes. Our algorithm only uses the portions of these spokes above ℓ, the dashed line.

4.3 Useful lemmas

Our algorithm relies on Lemmas 4, 5 and 6. These are stated below.

Recall that the contour also includes its portions below ℓ, i.e., it is defined with respect to the complete Voronoi diagram $\mathcal{VD}(S)$. We first have the following lemma, which is also needed in the proof of Lemma 5; a similar result on the standard Voronoi diagrams is already used in [9].

Lemma 4 *Each contour component never terminates or splits; that is, it has the topology of an infinite line or a circle.*

Proof. A contour component is made up of edges in $\mathcal{VD}(S)$, so if it were to terminate or split, it would be at a Voronoi vertex of $\mathcal{VD}(S)$. Due to our general position assumption that no point is equidistant to four sites, each Voronoi vertex in $\mathcal{VD}(S)$ is adjacent to three sites in S. If the contour hits a Voronoi vertex v, then at least one of these sites must be in S_a and at least one must be in S_b. Without loss of generality, let these sites be s_1, s_2, and s_3 with $s_1, s_2 \in S_a$ and $s_3 \in S_b$. The Voronoi edge between s_1 and s_3 and the Voronoi edge between s_2 and s_3 will be on the contour, so the contour will not terminate at v. The edge between s_1 and s_2 will not be on the contour, so the contour will not split at v. □

Lemma 5 *If a contour component contains a point above ℓ, then the contour component must intersect ℓ.*

Proof. Lemma 4 establishes that a contour component divides the plane into two regions, called *contour regions*. Notice that because a contour component is made up of edges in $\mathcal{VD}(S)$, each contour region must contain at least one Voronoi region of $\mathcal{VD}(S)$ and thus contains at least one site of S by Observation 1.

Now assume to the contrary that a contour component C contains a point above ℓ but C does not intersect ℓ. Then, the entire C is above ℓ. As such, one of the contour regions divided by C must be entirely in the halfplane above ℓ; let R be the region. This implies that the sites of S contained in R must be above ℓ, but this contradicts the fact that all sites of S are below ℓ. □

Lemma 6

1. *The total number of intersections between the contour and the Voronoi edges of the complete Voronoi diagrams $\mathcal{VD}(S_a)$ and $\mathcal{VD}(S_b)$ is at most $O(n)$.*
2. *The total number of intersections between the contour and the spokes of the complete Voronoi diagrams $\mathcal{VD}(S_a)$ and $\mathcal{VD}(S_b)$ is at most $O(n)$.*

Proof. We adapt the proof from [9] for a similar lemma on standard Voronoi diagrams.

Notice that the intersection between the contour and a Voronoi edge in $\mathcal{VD}(S_a)$ or $\mathcal{VD}(S_b)$ is a vertex in $\mathcal{VD}(S)$. There are $O(n)$ vertices in $\mathcal{VD}(S)$, so the total number of intersections between the contour and the Voronoi edges of $\mathcal{VD}(S_a)$ and $\mathcal{VD}(S_b)$ is at most $O(n)$. This proves the first lemma statement.

To prove the second lemma statement, we show that the contour can intersect each spoke at most once. We exploit the fact that Voronoi regions are star-shaped (Observation 1). If the contour intersects a spoke of the Voronoi region for site s in $\mathcal{VD}(S_a)$ or $\mathcal{VD}(S_b)$, then the open segment between s and this intersection will lie in the Voronoi region of s in $\mathcal{VD}(S)$. Because this segment is in the Voronoi region for s in $\mathcal{VD}(S)$, the contour cannot intersect this segment.

Now, assume for the sake of contradiction that the contour were to intersect a spoke twice. This would mean the closer to s of the two intersections would lie on the segment between s and the further of the two intersections, which we have shown above to be impossible. Therefore, the contour can only intersect each spoke at most once, and there are $O(n)$ spokes in $\mathcal{VD}(S_a)$ and $\mathcal{VD}(S_b)$, so the total number of intersections between the contour and the spokes is at most $O(n)$. □

5 Algebraic decision tree algorithm

Under the algebraic decision tree model, where the time complexity is measured only by the number of comparisons, we show that the IOAWNN-SL problem can be solved using $O(n \log n)$ comparisons. Consequently, we can solve the shortest path problem in weighted unit-disk graphs in $O(n \log n)$ time under the algebraic decision tree model. In the following, we first describe an $O(n \log^2 n)$ time algorithm under the conventional computational model and then show how to improve it to $O(n \log n)$ time under the algebraic decision tree model.

Let $p_1, p_2, \ldots, p_n$ be the points to be inserted in this order; each point has a weight. Let P denote the set of all these points. Let Q be a set of $O(n)$ query points, such that all points of P are above the x-axis ℓ while all points of Q are below ℓ. For each query point $q \in Q$, we know the timer when the query is conducted, i.e., we know the index i such that the query looks for the nearest neighbor of q among the first i points of P. Our goal is to answer all queries for the points of Q.

We construct a complete binary tree T whose leaves from left to right correspond to points $p_1, p_2, \ldots, p_n$ in this order. For each node $v \in T$, let P_v denote the set of points that are in the leaves of the subtree rooted at v. Let $\mathcal{VD}(P_v)$ be the additively-weighted Voronoi diagram for the weighted points of P_v; let $\mathcal{VD}_+(P_v)$ be the portion of $\mathcal{VD}(P_v)$ above ℓ. We construct $\mathcal{VD}_+(P_v)$. If we construct $\mathcal{VD}_+(P_v)$ for all nodes v of T in a bottom-up manner and use our linear time merge algorithm in Theorem 2, constructing the diagrams $\mathcal{VD}_+(P_v)$ for all nodes $v \in T$ can be done in $O(n \log n)$ time. In ad-

dition, we construct Kirkpatrick's point location data structure [10] on $\mathcal{VD}_+(P_v)$ for each node $v \in T$, which takes $O(|P_v|)$ time. [1] Note that we use Kirkpatrick's point location data structure instead of others such as the one in [6] because we will need to apply a technique from [5] that requires Kirkpatrick's data structure. Constructing the point location data structures for all nodes of T takes $O(n \log n)$ time.

Consider a query point $q \in Q$. Suppose we are looking for the nearest neighbor of q among the first i points $p_1, p_2, \ldots, p_i$ of P. Let v_i be the leaf of T corresponding to p_i. Following the path in T from the root to v_i, we can find a set V_q of nodes of T such that the union of P_v for all $v \in V_q$ is exactly $\{p_1, p_2, \ldots, p_i\}$. As such, the query can be answered after performing $O(\log n)$ point location queries on $\mathcal{VD}_+(P_v)$ for all $v \in V_q$. As each point location query takes $O(\log n)$ time, answering the nearest neighbor query for q can be done in $O(\log^2 n)$ time. Therefore, the total time for answering the queries for all points of Q is $O(n \log^2 n)$.

The above solves the problem in $O(n \log^2 n)$ time. To improve the time to $O(n \log n)$, the bottleneck is to solve all $O(n \log n)$ point location queries. For this, we resort to a technique recently developed by Chan and Zheng [5] under the algebraic decision tree model. We can simply apply [5, Theorem 7.2] to solve all our point location queries using $O(n \log n)$ comparisons (specifically, following the notation in [5, Theorem 7.2], we have $t = O(n)$, $L = O(n \log n)$, $M = O(n \log n)$, and $N = O(n)$ in our problem; according to the theorem, all point location queries can be solved using $O(L + M + N \log N)$ comparisons, which is $O(n \log n)$). Note that the theorem statement requires the input planar subdivisions to be triangulated. The triangulation is mainly used to construct Kirkpatrick's point location data structure [10] on each planar subdivision. Since we already have Kirkpatrick's point location data structure for each $\mathcal{VD}_+(P_v)$ as discussed above, we can simply follow the same algorithm of the theorem.

[1] Note that Kirkpatrick's data structure is originally for planar subdivisions in which each edge is a straight line segment. However, as discussed in [10], the algorithm also works for additively weighted Voronoi diagrams (and other types of Voronoi diagrams) since each cell of the diagram is star-shaped. A subtle issue in our problem is that $\mathcal{VD}_+(P_v)$ is only the portion of the complete diagram $\mathcal{VD}(P_v)$ above ℓ, and each cell of $\mathcal{VD}_+(P_v)$ does not contain its site. To circumvent the issue, we can enlarge each cell of $\mathcal{VD}_+(P_v)$ by including its site, as follows. For each cell $R \in \mathcal{VD}_+(P_v)$, if $\overline{ab}$ is a maximal segment of $R \cap \ell$, then we add the triangle $\triangle pab$ to R, where p is the site of R. Note that $\triangle pab$ must be inside the cell of p in $\mathcal{VD}(P_v)$, denoted by R'. As such, the enlarged region R is still star-shaped, contains its site p, and is a subset of R'. We can then construct Kirkpatrick's point location data structure on the subdivision of all these enlarged regions R.

References

[1] J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8:244–251, 1979.

[2] S. Cabello and M. Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry: Theory and Applications*, 48:360–367, 2015.

[3] T. M. Chan. Dynamic geometric data structures via shallow cuttings. *Discrete and Computational Geometry*, 64:1235–1252, 2020.

[4] T. M. Chan and D. Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016.

[5] T. M. Chan and D. W. Zheng. Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees. *ACM Transactions on Algorithms*, 2023.

[6] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

[7] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[8] H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discrete and Computational Geometry*, 64:838–904, 2020.

[9] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 18–27, 1979.

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

[11] C. Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM Journal on Computing*, 51:723–765, 2022.

[12] M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983.

[13] L. Roditty and M. Segal. On bounded leg shortest paths problems. *Algorithmica*, 59:583–600, 2011.

[14] M. I. Shamos and D. Hoey. Closest-point problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 151–162, 1975.

[15] H. Wang and J. Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020.

Grid-edge unfolding orthostacks with rectangular slabs*

Klára Pernicová†

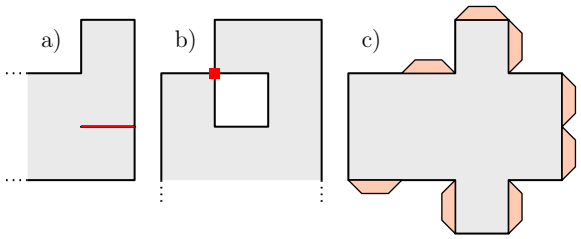


Figure 1: a) A part of a net with an overlapping edge. b) A part of a net with an overlapping vertex. c) A net resulting from a simple unfolding. There is enough space for glue regions.

Abstract

An orthostack with rectangular slabs is an orthogonal polyhedron obtained by stacking axis-parallel boxes on top of each other.

A grid-edge unfolding of an orthogonal polyhedron is obtained by cutting the surface of the polyhedron along segments in the intersection of axis-parallel planes passing through the vertices of the polyhedron and mapping the cut surface isometrically into the plane with no interior overlap.

We prove that orthostacks with rectangular slabs can be grid-edge unfolded into a simple polygon so that no faces, edges, or vertices overlap.

1 Introduction

A polyhedron is an *orthogonal polyhedron* if each face is parallel to an xy, yz, or xz plane.

Let $P \subseteq \mathbb{R}^3$ be an orthogonal polyhedron. Let $z_0, z_1, \ldots, z_n$ be all distinct z-coordinates of its vertices and assume that $z_0 < z_1 < z_2 < \cdots < z_n$. The *slab* S_i is the part of the polyhedron with z-coordinates between z_i and z_{i+1}, the *bottom face* of S_i is the subset of points of S_i with z-coordinate z_i, the *top face* of S_i is the subset of points of S_i with z-coordinate z_{i+1}. The orthogonal polyhedron P is called an *orthostack* if each slab is a prism whose base is a simple polygon. In this paper, we focus on orthostacks whose slabs are axis-parallel boxes and call them *orthostacks with rectangular slabs*.

*This work is supported by Project 23-04949X of the Czech Science Foundation (GAČR).

†Faculty of Mathematics and Physics, Charles University, klapernicova@gmail.com

We aim to obtain a special form of a planar net of a given orthostack P. We cut the surface of P along a subset of segments that are intersections of the surface of P with axis-parallel planes passing through the vertices of P. Then a *grid-edge unfolding* of P is an isometric mapping of the cut surface into the plane with no interior overlap. The image of the unfolding is called a *net*. A grid-edge unfolding is *simple* if no vertices (and therefore no edges) overlap; in other words, the images of the cutting segments form a simple closed curve after flattening to the net; see Figure 1. A net created by a simple unfolding is called a *simple net*. Simple nets are more practical for constructing polyhedra from paper as they leave some space for regions that can be used to add glue.

Our main result is the following.

Theorem 1 *Every orthostack with rectangular slabs has a simple grid-edge unfolding.*

To prove Theorem 1 we provide an algorithm for the unfolding in Section 3 and prove its correctness in Section 4.

1.1 Related results

A long-standing open question by Dürer asks whether every convex polyhedron can be edge-unfolded into a single simple polygon, where the cuts are allowed only along the edges of the polyhedron. The edge-unfolding does not exist for some non-convex orthogonal polyhedra, for example, a cube with a small hole in the middle of one face. We refer to survey papers by O'Rourke [10, 11, 12] for a broader overview.

The following three papers, which study grid-edge unfolding of orthostacks, are most related to our result.

Biedl et al. [1] proved that orthostacks could be unfolded allowing the cuts along segments from grid-edge unfolding and also along segments in horizontal planes with z-coordinates $(z_i + z_{i+1})/2$.

Damian and Meijer [7] studied orthostacks with orthogonally convex slabs. They found an algorithm for grid-edge unfolding of such orthostacks with an additional restriction stating that the boundary of each face within the top boundary of the slab S_i has two orthogonally incident edges that belong to the bottom boundary of the slab S_{i+1}; see Figure 2.

Chambers, Sykes, and Traub [2] showed that a grid-edge unfolding exists for a special class of orthostacks

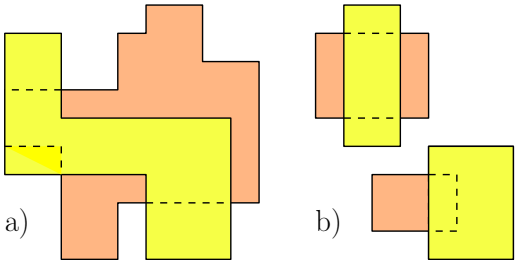


Figure 2: a) An orthostack with orthogonally convex slabs considered by Damian and Meijer [7]. b) Orthostacks with rectangular slabs that do not satisfy the requirements of Damian and Meijer [7].

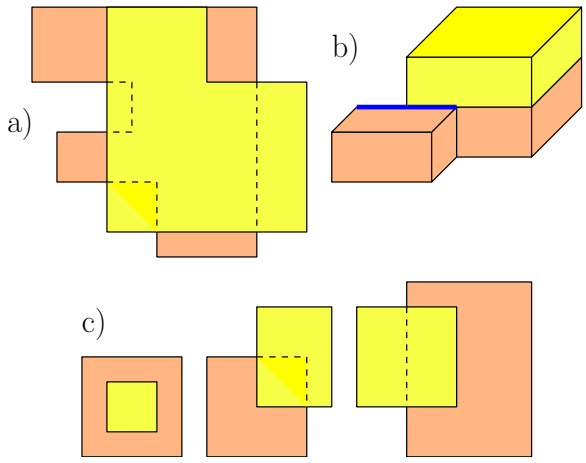


Figure 3: a) A top view of an orthostack satisfying both restrictions 1) and 2) considered by Chambers, Sykes, and Traub [2]. b) An orthostack violating restriction 2); the blue line highlights an edge that partially lies in the side boundary of the top slab and partially in the side boundary of the bottom slab. c) Top views of orthostacks with rectangular slabs violating restriction 1).

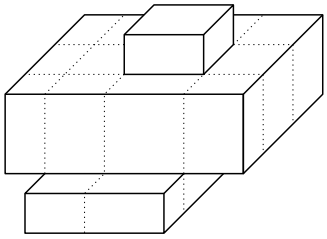


Figure 4: The new definition of faces for the purpose of grid-edge unfolding.

satisfying the following conditions; see Figure 3: 1) All top and bottom faces, except the top face of the topmost slab and the bottom face of the bottommost slab, are rectangles. 2) Every edge of every rectangular horizontal face lies completely within a side boundary (left, front, right, or back) of an adjacent slab.

If we allow arbitrary cuts on the surface of the polyhedron, then an unfolding exists for all convex polyhedra [9, Theorem 24.1.2].

Let P be an orthogonal polyhedron. As it is common in the literature we redefine the notion of a face as follows. We subdivide the surface of P with axis-parallel planes passing through the vertices of P. We will call the parts of the subdivision the *faces* of P. Note that in this definition each face of P is a rectangle. See Figure 4 for an illustration.

One could further subdivide each face of the polyhedron using an $a \times b$ orthogonal grid, allowing cuts along these grid lines. This process is termed a *refinement* and is characterized by the parameters a and b, which may also depend on the number of vertices of the polyhedron P. It has been demonstrated that all orthostacks can be unfolded using 1×2 refinement [1] and all genus-0 orthogonal polyhedra can be unfolded using various levels of refinement: exponential refinement [6], quadratic refinement [4], and linear refinement [3]. More recently, Damian, Demaine, Flatland, and O'Rourke have developed an unfolding method for all genus-2 orthogonal polyhedra using only linear refinement [5].

2 Notation

Faces parallel to the xy plane are called *horizontal*, faces parallel to the xz plane are called *front-back*, and faces parallel to the yz plane are called *left-right*.

Since each slab S_i is an axis-parallel box, we can express it as a Cartesian product

$$S_i = [x_{i,1}, x_{i,2}] \times [y_{i,1}, y_{i,2}] \times [z_i, z_{i+1}] .$$

Let E_i be the union of all the horizontal faces of P with z-coordinate z_i (it consists of the top faces of the slab S_{i-1} and the bottom faces of the slab S_i). Let L_i be the union of the left-right faces in S_i with x-coordinate

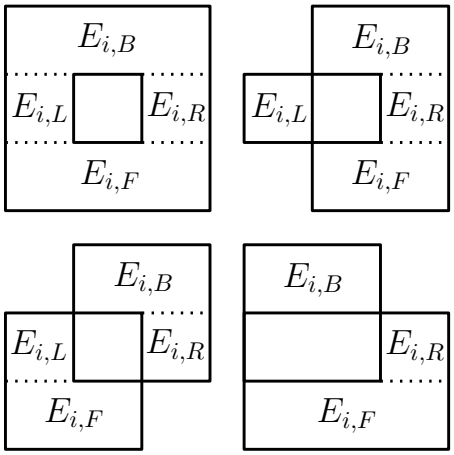


Figure 5: The rectangles are the projections to the xy plane of two adjacent slabs; the pictures do not distinguish which slab is above the other one. Dotted lines show the subdivision of E_i into $E_{i,F}$, $E_{i,R}$, $E_{i,B}$, $E_{i,L}$ and are the only places where we cut across an original face of the polyhedron. Pictures show only some cases of how two consecutive slabs can interact.

$x_{i,1}$ (it is the left rectangular boundary of S_i). Similarly, let R_i be the union of the left-right faces in S_i with x-coordinate $x_{i,2}$, let F_i be the union of the front-back faces in S_i with y-coordinate $y_{i,1}$, and let B_i be the union of the front-back faces in S_i with y-coordinate $y_{i,2}$. The surface of P is exactly the union of E_i for $0 \le i \le n$ and of L_i, R_i, F_i and B_i for $0 \le i < n$.

We subdivide E_i for $i \in \{1, 2, \ldots, n-1\}$, only E_0 and E_n remain untouched. We denote by $E_{i,L}$ the following subset of E_i (see Figure 5):

$$E_{i,L} = \{(x, y, z_i) \in E_i; \\ x \in [\min(x_{i-1,1}, x_{i,1}), \max(x_{i-1,1}, x_{i,1})] \\ \wedge\ y \in [y_{i-1,1}, y_{i-1,2}] \cap [y_{i,1}, y_{i,2}]\}.$$

Similarly we define $E_{i,R}$, $E_{i,F}$ and $E_{i,B}$:

$$E_{i,R} = \{(x, y, z_i) \in E_i; \\ x \in [\min(x_{i-1,2}, x_{i,2}), \max(x_{i-1,2}, x_{i,2})] \\ \wedge\ y \in [y_{i-1,1}, y_{i-1,2}] \cap [y_{i,1}, y_{i,2}]\},$$

$$E_{i,F} = \{(x, y, z_i) \in E_i; \\ y \in [\min(y_{i-1,1}, y_{i,1}), \max(y_{i-1,1}, y_{i,1})]\},$$

$$E_{i,B} = \{(x, y, z_i) \in E_i; \\ y \in [\min(y_{i-1,2}, y_{i,2}), \max(y_{i-1,2}, y_{i,2})]\}.$$

Note that $E_{i,L}, E_{i,R}, E_{i,F}, E_{i,B}$ are pairwise internally disjoint. Observe that each of these sets is either empty or a rectangle contained either in the top boundary of S_{i-1} or in the bottom boundary of S_i. The lines between $E_{i,L}, E_{i,R}, E_{i,F}, E_{i,B}$ are the only places where we cut across an original face of the polyhedron.

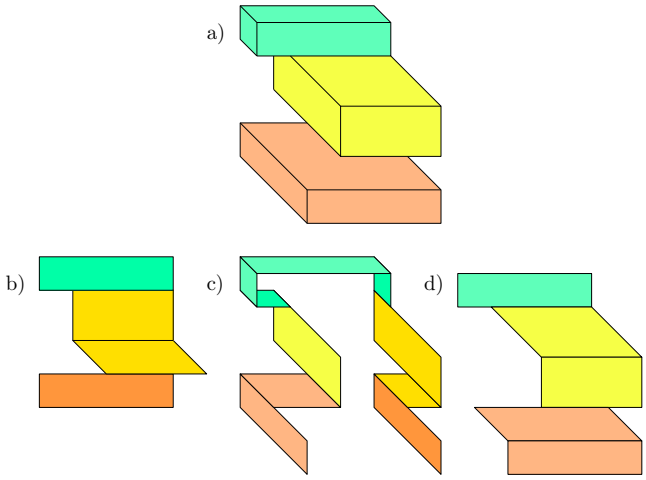


Figure 6: a) An orthostack with three rectangular slabs. b) All the faces unfolded during the back phase. c) All the faces unfolded during the right-left phase. d) All the faces unfolded during the front phase.

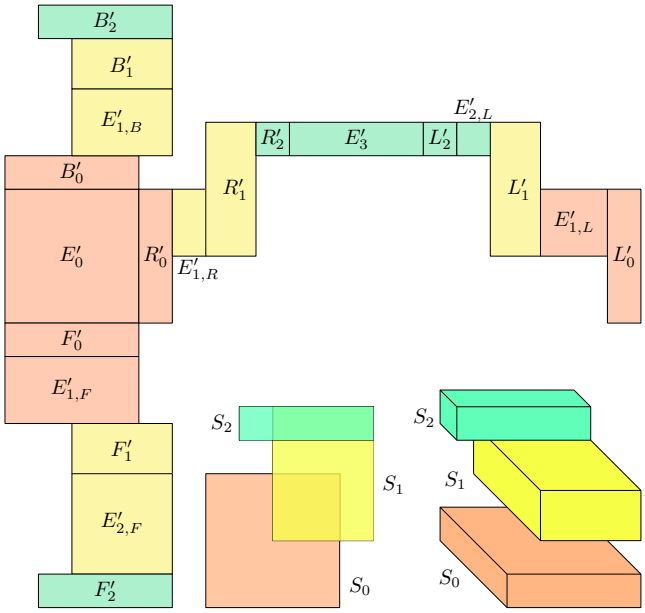


Figure 7: An orthostack with three rectangular slabs and its net resulting from the unfolding algorithm.

3 Unfolding algorithm

Let P be an orthostack with rectangular slabs. For a given subset A of the surface of P, we will denote by A' the corresponding subset in the constructed planar net.

We divide the algorithm into three phases, see Figure 6 for the division of faces into phases. We start with projecting E_0 orthogonally to the xy-plane. Each phase unfolds a part of the orthostack. See Figure 7 for the resulting net. The cutting segments will be clear from the process and described in detail in Section 4.

3.1 Right-left phase

In this phase we unfold all the rectangles R_i, $E_{i,R}$, L_i, $E_{i,L}$ and E_n. Let RE be the union of all the rectangles R_i and $E_{i,R}$, and let LE be the union of E_n and all the rectangles L_i and $E_{i,L}$.

We start with placing all rectangles from RE. We place R_0 to the right of E'_0, then $E_{1,R}$ to the right of R'_0, then R_1 to the right of $E'_{1,R}$, and we continue placing R_i and $E_{i,R}$, for $i = 2, 3, \ldots, n-1$, always to the right of the previous one. We then place to the right the whole rectangle E_n.

We continue placing the remaining rectangles from LE. We place $L_{n-1}, E_{n-1,L}, L_{n-2}, E_{n-2,L}, \ldots, L_0$ in this order, always to the right. Clearly, in the resulting net no vertices, edges, or faces overlap so far.

Note that the y-coordinates of the rectangles are preserved in this phase; they are the same in the orthostack and the net.

3.2 Back phase and front phase

Now we describe the second and the third phases. We denote the union of all the rectangles B_i and $E_{i,B}$ by BE and the union of all the rectangles F_i and $E_{i,F}$ by FE.

In the second phase, the back phase, we will be placing the rectangles of BE in the direction of increasing y-coordinate. We place B_0 above E'_0. Then we proceed with $E_{1,B}, B_1, E_{2,B}, B_2, \ldots, E_{n-1,B}, B_{n-1}$ in this order, placing each rectangle always above the previous one. The rectangle E_n has already been placed in the right-left phase.

In the third phase, the front phase, we will be placing the rectangles of FE in the direction of decreasing y-coordinate. We place F_0 below E'_0. Then we proceed with $E_{1,F}, F_1, E_{2,F}, F_2, ..., E_{n-1,F}, F_{n-1}$ in this order, placing each rectangle always below the previous one.

In the second and third phases, the x-coordinates of the rectangles are preserved.

4 Proof of non-overlap

Each phase on its own creates a simple non-overlapping polygon because of the continuous one-directional process. For a similar reason, rectangles from the back phase and the front phase cannot overlap. We will prove that no rectangle from the back phase can overlap or touch with a rectangle from the right-left phase. The proof for the rectangles from the front phase and the right-left phase would be analogous.

We will define a piecewise linear curve C formed by a subset of the cutting segments on the surface of P, which will partially separate rectangles from the right-left phase and the back phase, see Figure 8.

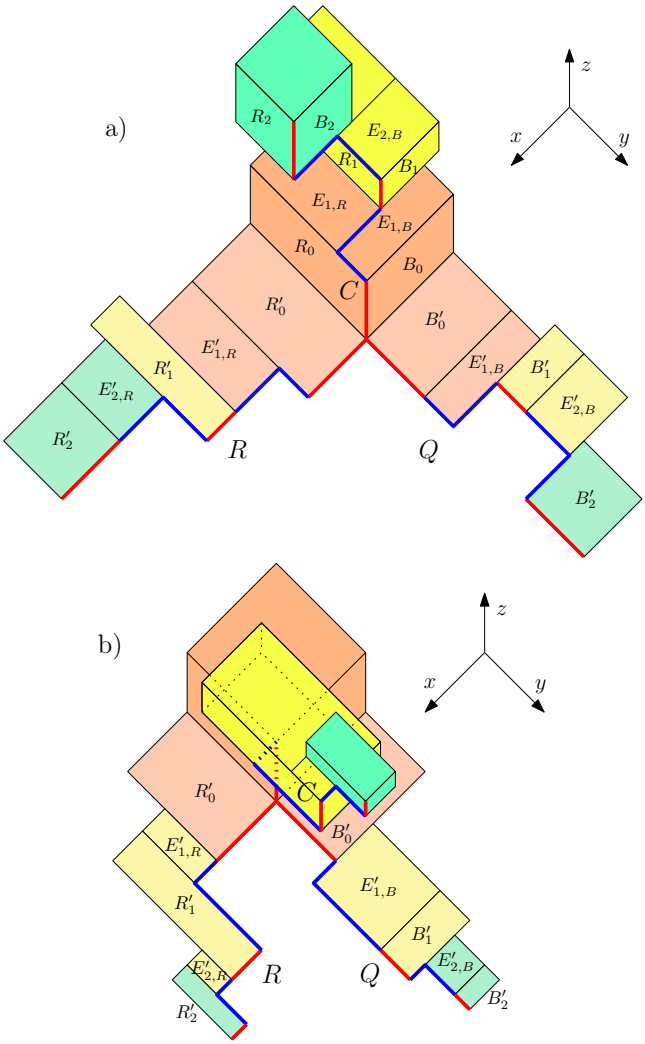


Figure 8: Orthostacks with a clear view of the right and the back part, the cut C and its images Q and R. In both pictures red lines separate B_i from R_i in the z direction and blue lines separate $B_i \cup E_{i+1,B} \cup B_{i+1}$ from $R_i \cup E_{i+1,R} \cup R_{i+1}$ in the x and y directions. a) Blue lines separate $E_{1,B}$ from R_0 in the y direction, $E_{1,B}$ from $E_{1,R}$ in the x direction, and then separate $E_{2,B}$ from R_1 in the y direction and B_2 from $E_{2,R}$ in the x direction. b) Blue lines separate $E_{1,B}$ from $E_{1,R}$ in the x direction, $E_{1,B}$ from R_1 in the y direction, and then separate B_1 from $E_{2,R}$ in the x direction and $E_{2,B}$ from R_2 in the y direction.

The curve C starts at $(x_{0,2}, y_{0,2}, z_0)$, which is the bottom right back corner of S_0, continues in the z direction to $(x_{0,2}, y_{0,2}, z_1)$, which is the top right back corner of S_0, then separates $B_0 \cup E_{1,B} \cup B_1$ from $R_0 \cup E_{1,R} \cup R_1$ in the x and y directions, reaching the point $(x_{1,2}, y_{1,2}, z_1)$, which is the bottom right back corner of S_1. The curve proceeds analogously until C reaches $(x_{n-1,2}, y_{n-1,2}, z_n)$, which is the top right back corner of the topmost slab S_{n-1}.

The curve C maps onto two piecewise linear curves in the net. One curve, denoted by Q, forms the right part of the perimeter of BE', and the second curve, denoted by R, forms the top part of the perimeter of RE'.

Our goal is to prove that the curves Q and R do not intersect.

When the curve C cuts between B_i and R_i, the corresponding part of the curve R moves in the increasing x-direction by $z_{i+1} - z_i > 0$ and the corresponding part of the curve Q moves in the increasing y-direction by $z_{i+1} - z_i > 0$. If C moves in the x-direction by d, then R moves to the right by d and Q moves to the left or right (preserving the former direction of C) by d. If C moves in the y-direction by d, then R moves up or down (preserving the former direction of C) by d and Q moves up by d.

The curves R and Q start diverging just after their common starting point when C cuts between B_0 and R_0: the curve R moves by $z_1 - z_0$ in the increasing x-direction and the curve Q moves by $z_1 - z_0$ in the increasing y-direction.

For every point A on the curve C we denote by A'_R the image of A on the curve R and we denote by A'_Q the image of A on the curve Q.

From the iterative process of constructing Q and R, we deduce the following.

Observation 1 *Let A be a point on the curve C except the starting point and assume that $A'_R = (x_R, y_R)$ and $A'_Q = (x_Q, y_Q)$. Then $x_R > x_Q$ and $y_R < y_Q$.* □

Lemma 2 *The curves R and Q do not intersect nor touch, except at the starting point $(x_{0,2}, y_{0,2})$.*

Proof. For a proof by contradiction suppose the curves Q and R intersect, see Figure 9. Denote by I' the point of the first intersection of Q and R (except the starting point). The point I' corresponds to a point U on C from the perspective of R and also corresponds to a point V on C from the perspective of Q (that is, $U'_R = V'_Q = I'$). By Observation 1 we have $U \neq V$. Assume, without loss of generality, that U appears on C before V. Then U'_Q is before V'_Q on Q. From Observation 1 the y-coordinate of U'_Q is greater than that of U'_R. Since Q is y-monotone and U is before V on C, the y-coordinate of U'_Q is smaller than or equal to the y-coordinate of $U'_R = V'_Q$. These two inequalities imply a contradiction. □

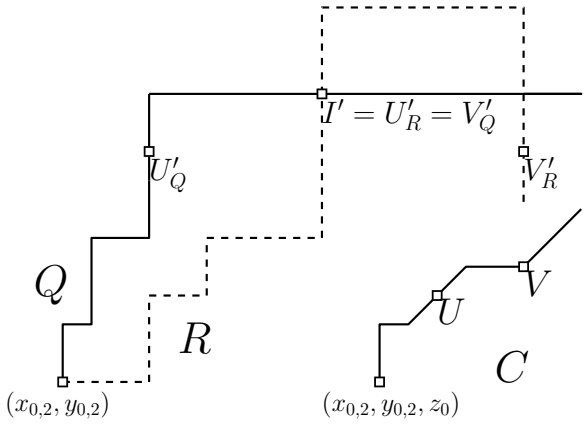


Figure 9: The cut C and vertices for proof.

The rectangles of BE' are to the left of the curve Q and the rectangles of RE' are below the curve R. Because the curve Q is to the left and upwards of the curve R and the curves Q and R do not intersect or touch, the rectangles of BE' and RE' cannot overlap.

Let S be a point on the curve C such that S'_Q is the rightmost point of Q. Then by Observation 1 the point S'_R is to the right of S'_Q. All the rectangles in LE' are to the right of S'_R, so the rectangles in BE' cannot overlap with the rectangles in LE'. This concludes the proof that rectangles placed in the back phase cannot overlap with the rectangles from the right-left phase.

5 Discussion

Our algorithm relies on connected faces within each phase and simple path cuts that separate the rectangles of different phases. It leaves no more space in the net between the back part and the right part, and between the right part and the front part; however, placing more faces next to the left part is possible. We plan to improve our algorithm so that each slab's left boundary does not have to consist of only one rectangle.

If we closely inspect the cutting segments of our algorithm, we can see that they form a single path. The path starts with the cut C separating the right part RE from the back part BE, followed by the back edge of E_n, a cut separating BE from LE, the left edge of E_0, a cut separating LE from FE, the front edge of E_n, and a cut separating FE from RE.

An edge-unfolding of a polyhedron that is obtained by cutting the surface along a path is called a *Hamiltonian unfolding* [13] or an *edge-unzipping* [12]. There exist orthogonal polyhedra with no edge-unzipping [8]. The characterization of orthogonal polyhedra that have edge-unzipping remains open.

References

[1] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, M. Overmars, J. O'Rourke, S. Robbins, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proc. 10th Canad. Conf. Comput. Geom.*, pages 70–71, 1998. Full version in *Elec. Proc.*: `http://www.cccg.ca/proceedings/1998/cccg98-biedl-unfolding.ps.gz`

[2] E. W. Chambers, K. A. Sykes, and C. M. Traub. Unfolding rectangle-faced orthostacks. In *Proc. 24th Canad. Conf. Comput. Geom.*, pages 23–27, 2012.

[3] Y.-J. Chang and H.-C. Yen. Improved Algorithms for Grid-Unfolding Orthogonal Polyhedra. *International Journal of Computational Geometry & Applications*, 27(01n02):33–56, 2017.

[4] M. Damian, E. D. Demaine, and R. Flatland. Unfolding orthogonal polyhedra with quadratic refinement: the delta-unfolding algorithm. *Graphs and Combinatorics*, 30(1):125–140, 2014.

[5] M. Damian, E. Demaine, R. Flatland, and J. O'Rourke. Unfolding genus-2 orthogonal polyhedra with linear refinement. *Graphs and Combinatorics*, 33(5):1357–1379, 2017.

[6] M. Damian, R. Flatland, and J. O'Rourke. Epsilon-unfolding orthogonal polyhedra. *Graphs and Combinatorics*, 23(1):179–194, 2007.

[7] M. Damian and H. Meijer. Grid edge-unfolding orthostacks with orthogonally convex slabs. *14th Annu. Fall Workshop Comput. Geom.*, pages 20–21, 2004.

[8] E. D. Demaine, M. L. Demaine, D. Eppstein, and J. O'Rourke. Some polycubes have no edge zipper unfolding. In *Proc. 32nd Canad. Conf. Comput. Geom.*, pages 101–105, 2020.

[9] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, 2007.

[10] J. O'Rourke. Unfolding orthogonal polyhedra. In J. E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, pages 307–317. Contemporary Mathematics, volume 453, American Mathematical Society, 2008.

[11] J. O'Rourke. Unfolding Polyhedra. 2008. Available online: `https://www.science.smith.edu/~jorourke/Papers/PolyUnf0.pdf`

[12] J. O'Rourke. Unfolding Polyhedra. In *Proc. 31st Canad. Conf. Comput. Geom.*, pages 85–86, 2019.

[13] G. C. Shephard. Convex polytopes with convex nets. *Math. Proc. Camb. Phil. Soc.*, 78:389–403, 1975.

On 3-layered Cornerhedra: Optimum Box Partitions for Niches

Laurie Heyer* William Lenhart† Ulrike Stege‡ Sue Whitesides‡

Abstract

We define a family of orthogonal polyhedra we call *niches*, which are certain unions of unit cubes (voxels) in an octant of the 3D integer lattice. We seek to partition the cubes into completely filled, interior-disjoint rectangular boxes using the smallest possible number of boxes. The number of extreme cubes, or peaks, provides a known lower bound for the number of boxes needed. We construct boxings (i.e., partitions into boxes) of optimum size, and characterize *perfect niches*, those niches for which optimum boxings achieve the lower bound.

1 Introduction

In the $3D$ lattice of grid points with non-negative integer coordinates, we consider orthogonal polyhedra that are unions of unit grid cells, or voxels, which we call *cubes*. A *box* is rectilinear, with 8 vertices at grid points, 6 rectangular faces, and 12 axis-aligned edges.

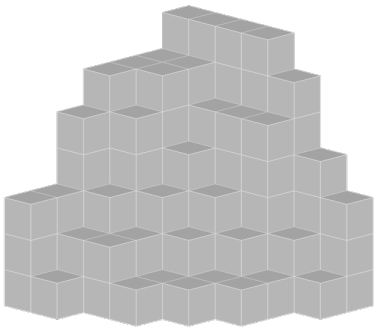


Figure 1: A cornerhedron with $k = 23$ corners, redrawn from Winkler's puzzle book [14].

A *cornerhedron* $\mathcal{C}$ (or corner polyhedron [2, 5]) is a union of cubes with the property that, for each cube c of $\mathcal{C}$, the smallest *box* containing c and the cube $c_{\mathcal{O}}$ at the origin is completely filled with cubes of $\mathcal{C}$ (no "air" inside the box). Any axis-aligned line intersects $\mathcal{C}$ in a single segment with one endpoint on a coordinate plane. See Figure 1 and Figure 3a, which we call a *triskele*.

*Mathematics & Computer Science, Davidson College, laheyer@davidson.edu

†Department of Computer Science, Williams College, wlenhart@williams.edu

‡Department of Computer Science, University of Victoria, {ustege,sue}@uvic.ca

We partition the cubes of cornerhedron $\mathcal{C}$ into a set of interior-disjoint boxes completely filled with cubes. We call such a partition a *boxing* $\mathcal{B}$ of $\mathcal{C}$, and we seek boxings that minimize the size $\|\mathcal{B}\|$ of $\mathcal{B}$.

A vertex of $\mathcal{C}$ is called a *peak* [2] if it is a local maximum in the $(1, 1, 1)$ direction. If a cube has a peak vertex, we call the cube a *corner*. We denote by k the number of corners (equivalently, the number of peaks) of $\mathcal{C}$.

As no two corners or peaks can belong to the same box, any boxing $\mathcal{B}$ must have at least as many boxes as the number k of corners of $\mathcal{C}$. If $\mathcal{C}$ has a boxing $\mathcal{B}$ that achieves this lower bound, i.e., $\|\mathcal{B}\| = k$, then we say that $\mathcal{C}$ and $\mathcal{B}$ are *perfect*.

Motivating examples. When cornerhedron $\mathcal{C}$ has just one layer (i.e., has height 1), then $\mathcal{C}$ is easily seen to be perfect, which also follows from work on partitioning rectilinear polygons by Dielissen and Kaldewaij [4]. When cornerhedron $\mathcal{C}$ has exactly two layers, we prove it is perfect in Theorem 2. For an example, see the 2-layer cornerhedron with 14 corners in Figure 4a and its perfect boxing $\mathcal{B}$ with $\|\mathcal{B}\| = 14$. When cornerhe-

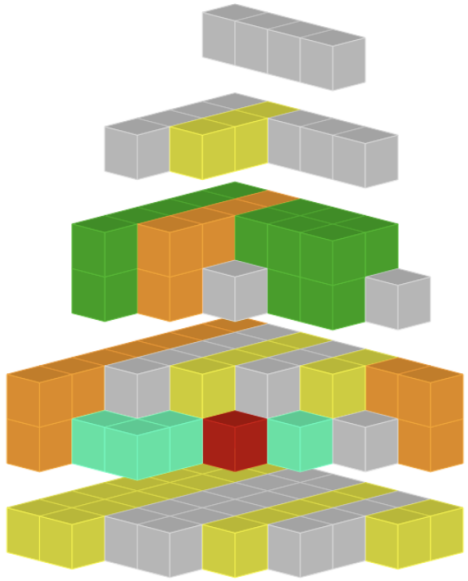


Figure 2: A perfect boxing $\mathcal{B}$ consisting of $\|\mathcal{B}\| = 23$ boxes of the (redrawn) cornherdron depicted on the cover of Winklers puzzle book [14].

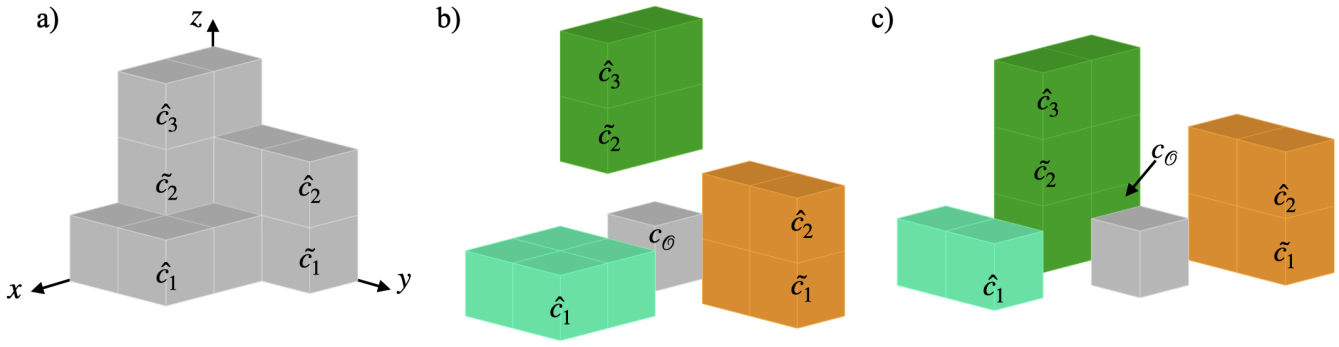


Figure 3: a) triskele; b) and c) exploded views of two boxings of the triskele.

dron $\mathcal{C}$ has exactly three layers, the *triskele* (Figure 3a) provides a fascinating instance. It has 13 cubes, including three corners, $\hat{c}_1$, $\hat{c}_2$, and $\hat{c}_3$. Hence at least three boxes are required to pack up the cubes of the triskele. We prove in Section 4 that the triskele requires at least *four* boxes. Hence the triskele is not perfect, and both boxings in Figure 3 are optimum. We claim that the cornerhedron with 23 peaks shown in Figure 1 is perfect; Figure 2 shows the 23 boxes of our perfect boxing of it.

Questions arising. What properties do imperfect 3-layer cornerhedra share? Can we characterize perfect 3-layer cornerhedra in a way that would lead to an efficient recognition algorithm? If a 3-layer cornerhedron is not perfect, can we at least find an optimum boxing for it, minimizing the number of boxes used?

Our main results concern a subfamily of 3-layer cornerhedra we call *niches*. In order to define niches and to state our main results, we answer the first of the above questions, after giving some notation.

Cubes. We identify a unit cube $c_{x,y,z}$ by the x, y, z-coordinates (in a right-handed coordinate system) of the vertex farthest from the origin $\mathcal{O} = (0, 0, 0)$. The cube $c_{1,1,1}$ with one vertex at $\mathcal{O}$ is called the *origin cube*, denoted $c_{\mathcal{O}}$ (see Figure 3b and c).

Layer-corners. Whether a cornerhedron $\mathcal{C}$ has one or many layers, each layer $z = i$ has at least one *layer-corner*: a cube $c_{x,y,i}$ of layer i such that neither $c_{x+1,y,i}$ nor $c_{x,y+1,i}$ exists, i.e., $c_{x,y,i}$ is an extreme cube in both the x- and y-directions. Layer-corners may or not be corners of $\mathcal{C}$. When a layer-corner has a cube on top of it, we call it a *hidden layer-corner*. We often denote hidden layer-corners by $\tilde{c}_i$, and layer-corners that are corners of $\mathcal{C}$ by $\hat{c}_i$. We refer to corners of $\mathcal{C}$ that lie in layer i as $\hat{i}$-*corners*. They are both corners of $\mathcal{C}$ and layer-corners of layer i. The triskele in Figure 3a has two layer-corners $\hat{c}_1 = c_{3,2,1}$ and $\tilde{c}_1 = c_{1,3,1}$ in layer $z = 1$ (its lowest layer), two layer-corners $\hat{c}_2 = c_{1,3,2}$ and $\tilde{c}_2 = c_{2,1,2}$ in layer 2, and one layer-corner $\hat{c}_3 = c_{2,1,3}$ in layer 3. The cornerhedron in Figure 4 ahead has three hidden layer-corners: $\tilde{c}_{x_1,y_1,1}$, $\tilde{c}_{x_2,y_2,1}$, and $\tilde{c}_{x_3,y_3,1}$.

Towers. A hidden layer-corner $\tilde{c}_1$ in layer 1 and the cube above it together build a *short tower* $\mathcal{S}$. A hidden layer-corner $\tilde{c}_2$ in layer 2 and the cube above it together build a *tall tower* $\mathcal{T}$ provided that layer 1 contains a *guard* that serves as a *witness*, i.e, a corner with x- and y-coordinates strictly greater than those of $\tilde{c}_2$. The triskele in Figure 3a has one short tower $(\tilde{c}_1 \cup \hat{c}_2)$ and one tall tower $(\tilde{c}_2 \cup \hat{c}_3)$, with $\hat{c}_1$ as a witness. The triskele (or its mirror image) is the smallest cornerhedron that has both a short and a tall tower.

With these definitions, we can now address our first question with a theorem that motivates the definition of niches. The definition follows the theorem.

Theorem 1 *Any cornerhedron $\mathcal{C}$ that has 3 layers but is not perfect has at least one short tower and at least one tall tower.*

Proof. Let $\mathcal{C}'$ denote the subcornerhedron of $\mathcal{C}$ formed by layers 2 and 3. If $\mathcal{C}$ has no short tower, then all layer-corners of layer 1 are corners. To box $\mathcal{C}$, we take the boxes of a perfect boxing for layer 1 and the boxes of an optimum boxing for $\mathcal{C}'$. Since $\mathcal{C}'$ is perfect (Theorem 2 ahead), this gives a perfect boxing of $\mathcal{C}$.

Likewise, we can box $\mathcal{C}$ perfectly if it has no tall tower. In this case, all layer-corners of layer 2 are corners, so we take the boxes of a perfect boxing for layer 3 together with the boxes of an optimum boxing for the first two layers, which is a perfect subcornerhedron. This gives a perfect boxing for $\mathcal{C}$. □

Niches. Based on Theorem 1, we define a *niche* $\mathcal{N}$ to be a 3-layer cornerhedron $\mathcal{C}$ that, like the triskele, has exactly one short tower and exactly one tall tower, such that one of the towers lies against the $x = 0$ plane and the other against the $y = 0$ plane; there are no additional towers. Niches can be perfect (Figures 6–10). Thus the family of niches, defined by two properties that every 3-layer imperfect cornerhedron must have, is the

smallest subfamily of cornerhedra where characterizing and recognizing perfection arises.

Main contributions. Theorem 3, stated in Section 2, characterizes perfect niches with a necessary and sufficient condition involving corners and their alignment properties. The proof of its sufficiency (Section 3) uses what we call our *anchored layer boxing* (ALB)-construction method, defined further on. The proof of its necessity (Section 4) uses what we call our $box(c_{\mathcal{O}})$ method for analysing a given boxing. Theorem 4 gives constructions for optimum boxings of niches. We believe that the concept of niches, and our ALB and $box(c_{\mathcal{O}})$ methods can contribute to resolving many combinatorial and complexity questions for 3-layer cornerhedra and possibly for cornerhedra in general.

Related work. In 1991, it was shown that the problem of partitioning an orthogonal polyhedron into a minimum number of boxes is NP-complete in $3D$, while solvable in polynomial time in $2D$ [4].

In 2018, Biedl et al. [2] considered the special case of partitioning 3D-histograms into a minimum number of boxes: the problem is NP-hard even for histograms of height two. For partitioning what we call cornerhedra here, they described a 2-approximation algorithm, and they posed the problem of determining whether the partitioning problem is polynomial or–at least–whether there exists a PTAS.

Floderus et al. gave a 4-approximation for partitioning histograms and applied their result to matrix multiplication [7].

In two dimensions, a number of papers have given polynomial time algorithms for partitioning a rectilinear polygon into the minimum possible number of rectangles. See Lipski et al. 1979 [10], Ohtsuki 1982 [13], and Ferrari et al. 1984 [6].

The representation of $3D$ objects as unions of cubes (voxels) arises in other diverse areas of theory and application. We give several examples below.

In combinatorial mathematics, the object we call a cornerhedron arises as a representation by unit cubes of a combinatorial object called a *plane partition*, defined by P.A. MacMahon [12]. A plane partition is a finite 2-dimensional array of non-negative integers such that the entries in each row are non-increasing as the column index increases (i.e., $a_{i,j} \geq a_{i,j+1}$), and likewise the entries in each column are non-increasing as the row index increases (i.e., $a_{i,j} \geq a_{i+1,j}$). A plane partition can be represented by unit cubes. One regards each entry $a_{i,j}$ of the 2-dimensional array as the number of unit cubes to be stacked at the cell labelled (i, j). This representation is a cornerhedron, and any cornerhedron gives rise to a plane partition. The main goal in the literature on plane partitions and their various classes is to count the number of plane partitions in the given class as a function of the number n of unit cubes in the representation.

Agarwal et al. [1] investigated a problem arising, e.g., in motion planning, asking whether the complement of a 3D object consisting of a collection of axis-aligned cubes can be partitioned into a collection of axis-aligned boxes.

Eppstein and Mumford [5] characterized which graphs can be drawn on the skeletons of corner polyhedra, which we call cornerhedra here (corner polyhedra of a different kind were defined by Gomory [8] in the context of linear programming).

When 3D printing a 3D object, it can be necessary to partition a model of the object into printable parts that are then assembled; see, e.g., Livisu et al. 2017 [11]. Similarly, a strategy for packing an object into a box is to break the object into pieces for packing and later reassembly; see, e.g., Chekanin 2020 [3].

Finally, as the cover of Winkler's book [14] suggests, cornerhedra are fascinating objects and suggest many puzzles and problems.

2 Preliminaries

We begin with notation. Then, to introduce our *anchored layer-boxing* (ALB) method by way of an example, we use it to prove, in Theorem 2, that all 2-layer cornerhedra are perfect. We state our characterization of perfect niches in Theorem 3. Based on Theorems 2 and 3, we then prove in Theorem 4 that an optimum boxing of a niche has k boxes if the niche is perfect, and $k + 1$ boxes if it is not perfect.

Recalling the definitions of niche N, short tower $\mathcal{S}$, and tall tower $\mathcal{T}$, from now on we assume (WLOG) that $\mathcal{S}$ lies against the plane $x = 0$ and that $\mathcal{T}$ lies against the $y = 0$ plane as shown in Figure 3a.

Layer i of cornerdron $\mathcal{C}$ consists of the cubes of $\mathcal{C}$ between two horizontal planes $z = i - 1$ and $z = i$, $i \geq 1$. Cubes in layer i have z-coordinate $z = i$; layer i and its cubes have *height* i. A box with cubes in exactly one, two, or three layers is called *thin*, *thick*, or *deep*, respectively. The boxing in Figure 3b has two thin boxes, two thick boxes, and no deep box; the boxing in Figure 3c has two thin boxes, one thick box, and one deep box.

A V-layer consists of the cubes of cornerhedron $\mathcal{C}$ between two vertical planes $x = i-1$ and $x = i$, or between $y = i - 1$ and $y = i$, where $i \geq 1$. Each cube c_{x_0,y_0,z_0} of $\mathcal{C}$ belongs to two V-layers, the V-layer between planes $x = x_0 - 1$ and $x = x_0$, and the V-layer between planes $y = y_0 - 1$ and $y = y_0$. A V-layer that contains corners of layers 2 and 3 is called a $V_{\hat{3}\hat{2}}$-layer, and similarly for other subsets of $\{\hat{3}, \hat{2}, \hat{1}\}$. Some corners in a V-layer may not appear in the subscript: a $V_{\hat{2}\hat{1}}$-layer might also be a $V_{\hat{3}\hat{2}\hat{1}}$-layer.

The V-layer between $x = 0$ and $x = 1$ is called the

$\mathcal{S}$-wall; the V-layer between $y = 0$ and $y = 1$ is the $\mathcal{T}$-wall. By assumption for niches, they contain $\mathcal{S}$ and $\mathcal{T}$, respectively. A V-layer that is perpendicular to the $\mathcal{T}$-wall or to the $\mathcal{S}$-wall is called a $V_{\mathcal{T}^\perp}$-layer or a $V_{\mathcal{S}^\perp}$-layer, respectively.

Given a corner $\hat{c}$ in layer 3, we call the cube in layer 1 whose x- and y-coordinates are 1 greater than those of $\hat{c}$ the *key* of $\hat{c}$. The key may be filled by a cube $c_{key}(\hat{c})$ of $\mathcal{C}$ ($\hat{c}$ is *keyed*) or it may be empty ($\hat{c}$ is *keyless*).

We often denote the corner at the top of a tall tower $\mathcal{T}$ by $\hat{c}_{\mathcal{T}}$. By definition of tall tower $\mathcal{T}$, layer 1 contains a guard corner for $\mathcal{T}$, which implies that $c_{key}(\hat{c}_{\mathcal{T}})$ exists.

Recalling the definition of layer-corner, we define below a particular perfect boxing of a single layer, the *anchored layer-boxing* (ALB). The single layer may be all of the cornerhedron, or it may occur as just one layer. We often construct perfect boxings by our ALB method: we make an ALB for each layer of the given cornerhedron to obtain an initial boxing and then modify the boxing if necessary. For the definition of anchored layer-boxing (ALB) below, refer to Figure 4b.

Definition [**ALB** *anchored layer-boxing*]. Given a layer at height $z \geq 1$ of a cornerhedron $\mathcal{C}$, let the coordinates of the layer-corners be indexed (x_i, y_i, z), where x_1 is the smallest of the x-coordinates and y_1 is the largest of the y-coordinates. We choose one layer-corner $c^*_{x_{i_0},y_{i_0},z}$ to be the *anchor* of the layer-boxing, and define the box containing it to be the box that contains the two cubes $c^*_{x_{i_0},y_{i_0},z}$ and $c_{1,1,z}$. We denote this box by $box(c^*_{x_{i_0},y_{i_0},z})$. The other boxes of the ALB are defined as follows. Each $box(c_{x_i,y_i,z})$ such that $x_i > x_{i_0}$ extends to the $y = 0$ plane and to the plane $x = x_{i-1}$. Each $box(c_{x_i,y_i,z})$ s.t. $x_i < x_{i_0}$ (i.e., $y_i > y_{i_0}$) extends to the $x = 0$ plane and to the plane $y = y_{i+1}$.

The proof of our next result serves to introduce the ALB construction method by way of an example. We will use this method extensively in Section 3, which gives a constructive proof of the sufficiency of our characterization of perfect niches. For the proof of the following theorem, recall the definition of hidden layer-corner.

Theorem 2 *Every 2-layer cornerhedron $\mathcal{C}$ is perfect.*

Proof. Let $\tilde{k}$ denote the number of hidden layer-corners, and let their coordinates be denoted $(x_i, y_i, 1)$, where $1 \leq i \leq \tilde{k}$; here x_1 is the minimum of the x_i, and y_1 is the maximum of the y_i (the index i increases with increasing distance from the plane $x = 0$). We choose $\tilde{c}_{x_1,y_1,1}$ and the corner $\hat{c}_{x_1,y_i,2}$ above it as anchors for ALBs of layers 1 and 2. We refer to the resulting boxes as the *thin* boxes. When $\tilde{k} = 0$ the thin boxes form a boxing $\mathcal{B}$ that is perfect. If $\tilde{k} > 0$ the set of thin boxes does not form a perfect boxing for cornerhedron $\mathcal{C}$ as the boxes of hidden layer-corners do not contain corners of $\mathcal{C}$. Thus we modify this set of thin boxes as follows.

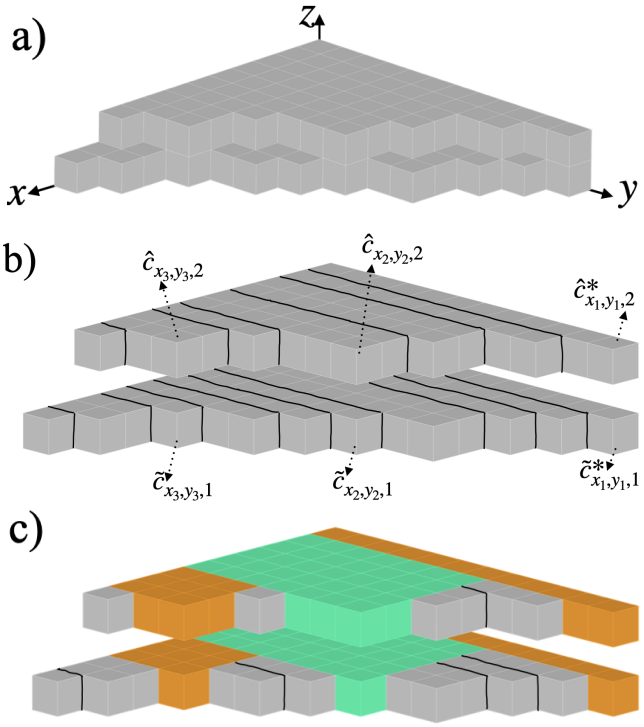


Figure 4: a) a 2-layer cornerhedron with $k = 14$ corners; b) ALBs anchored at $\hat{c}^*_{x_1,y_1,2}$ and $\tilde{c}^*_{x_1,y_1,1}$; and c) a perfect boxing with 14 boxes. The three thick boxes, in color, contain the hidden layer-corners.

We first thicken the thin box of $\hat{c}^*_{x_1,y_1,2}$ by combining it with the thin box of $\tilde{c}^*_{x_i,y_i,1}$ below it. Because these anchors have the same x, y-coordinates, their thin boxes have the same footprint on plane $z = 0$. Thus the thick box contains the cube $c_{\mathcal{O}}$, and the remaining thin boxes are not affected.

When $\tilde{k} > 1$, for $1 < i \leq \tilde{k}$, we thicken the box of $\hat{c}_{x_i,y_i,2}$ so that the box contains $\tilde{c}_{x_i,y_i,1}$. We extrude this thick box to the thick box of $\hat{c}_{x_{i-1},y_{i-1},2}$, i.e., to the plane $x = x_{i-1}$. Extruding the thick box for $\hat{c}_{x_i,y_i,2}$ truncates the thin boxes of non-hidden layer-corners (i.e., corners) of layers 1 and 2 having x-coordinates strictly between x_{i-1} and x_i: these thin boxes no longer abut the plane $y = 0$ but instead abut a face of the extruded thick box for $\hat{c}_{x_i,y_i,2}$ on plane $y = y_i$.

Modifying the set of thin boxes as described gives a perfect boxing for $\mathcal{C}$. □

To obtain the perfect boxing $\mathcal{B}$ shown in Figure 4c, thick boxes (in color) are created for the hidden layer-corners, in order of increasing distance from the $x = 0$ plane. The subcornerhedra (shown grey) have no hidden layer-corners and so are perfect. The thick boxes and the thin grey boxes together form a perfect boxing of size k.

Next we state our necessary and sufficient condition for a niche to be perfect. Sufficiency is proved in Section 3 and necessity in Section 4.

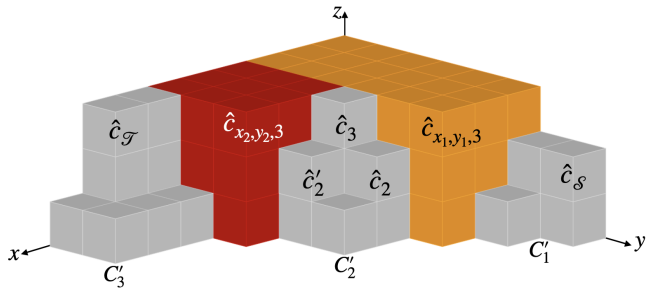


Figure 5: For Lemma 5: $k^- = 2$; $\mathcal{C}'_1$, $\mathcal{C}'_2$, $\mathcal{C}'_3$ (right to left, grey). Large deep box (orange) contains keyless $\hat{3}$-corner $\hat{c}_{x_1,y_1,3}$. Small deep box (red) contains $\hat{c}_{x_2,y_2,3}$.

Theorem 3 (characterization of perfect niches) *A niche $\mathcal{N}$ is perfect if and only if $\mathcal{N}$ has at least one of the following features:*
i) a $V_{\hat{3}\hat{1}}$-layer containing $\hat{c}_{\mathcal{T}}$; or
ii) a $V_{\hat{2}\hat{1}}$-layer perpendicular to the $\mathcal{S}$-wall; or
iii) a $V_{\hat{3}\hat{2}}$-layer perpendicular to the $\mathcal{T}$-wall; or
iv) a $\hat{3}$-corner whose V-layers together contain a $\hat{2}$-corner and a $\hat{1}$-corner; or
v) a keyless $\hat{3}$-corner.

We refer to the condition of Theorem 3 simply as the *Condition*. A niche $\mathcal{N}$ may have none, or some, of *features i)-v)* of the Condition. Two corners *align* if they belong to the same V-layer. Features i)-iv) describe alignments of corners. These alignments are crucial to the ALB-constructions in the proof of sufficiency.

Theorem 4 (optimum boxing for niches) *Every niche $\mathcal{N}$ with k corners has an optimum boxing $\mathcal{B}$ such that $\|\mathcal{B}\| = k$ when $\mathcal{N}$ is perfect and such that $\|\mathcal{B}\| = k+1$ when $\mathcal{N}$ is not perfect.*

Proof. When $\mathcal{N}$ is perfect, then by Theorem 3, it has at least one of features i)-v). A perfect boxing $\mathcal{B}$ corresponding to each feature is given in the proofs of Lemmas 5 and 6 in Section 3. When $\mathcal{N}$ is not perfect, i.e., when it fails to have any of the features i)-v), then we obtain an optimum boxing $\mathcal{B}$ as follows.

We box the cubes of the $\mathcal{T}$-wall into two boxes, a box b_1 containing $\hat{c}_{\mathcal{T}}$ and $c_{\mathcal{O}}$, and a box b_2 containing the remaining cubes of the $\mathcal{T}$-wall. These remaining cubes are all in layer 1 and include among them the cube $c_{x_{max},1,1}$ with maximum x-coordinate in the $\mathcal{T}$-wall. While this cube in box b_2 is extreme in the x-direction, it is not a corner because $\mathcal{N}$, being imperfect, cannot have feature i). Then we remove the $\mathcal{T}$-wall from $\mathcal{N}$, leaving a subcornerhedron $\mathcal{N}'$ with $k-1$ corners and only one tower, $\mathcal{S}$. By Theorem 1, $\mathcal{N}'$ has a perfect boxing $\mathcal{B}'$ (obtained by the ALB-method in Theorem 2), which together with boxes b_1 and b_2, gives a boxing $\mathcal{B}$ of $\mathcal{N}$. Since $\|\mathcal{B}\| = k+1$ and $\mathcal{N}$ is not perfect, $\mathcal{B}$ is an optimum boxing. □

3 Sufficiency: Boxing Perfect Niches Perfectly

First we prove that a niche having feature v) of Theorem 3 is perfect. The proof considers keyless $\hat{3}$-corners in order of increasing distance from plane $x = 0$, the same order in which the proof of Theorem 2 considered hidden layer-corners of a 2-layer cornerhedron.

Lemma 5 *Niche $\mathcal{N}$ is perfect if it has at least one keyless $\hat{3}$-corner.*

Proof. Let k^- denote the number of keyless $\hat{3}$-corners of $\mathcal{N}$, and let their coordinates be denoted $(x_i, y_i, 3)$, where $1 \le i \le k^-$; x_1 is the minimum of the x_i, and y_1 is the maximum of the y_i. To create a perfect boxing $\mathcal{B}$ for $\mathcal{N}$, we first create a deep box for each keyless $\hat{3}$-corner $\hat{c}_{x_i,y_i,3}$ as shown in Figure 5 for $k^- = 2$. The deep box for $\hat{c}_{x_1,y_1,3}$, shown orange, contains $c_{\mathcal{O}}$. The deep box for each subsequent keyless $\hat{3}$-corner extends to the plane $y = 0$ and to the side of the previous deep box. This creates k^-+1 subcornerhedra $\mathcal{C}'_1, \ldots, \mathcal{C}'_{k^-+1}$, shown grey. Each $\mathcal{C}'_i$ is perfect, whether it is empty, or has height 1, or height 2 by Theorem 2, or height 3 by Theorem 1. In any case, the ALB method of Theorem 2 applies. The perfect boxings of the $\mathcal{C}'_i$ together with the deep boxes form a perfect boxing $\mathcal{B}$ for $\mathcal{N}$. □

Having shown the sufficiency of feature v) in Lemma 5, our strategy for proving the sufficiency of each of the remaining features i)-iv) is this: we make an ALB for each layer, using the corners of $\mathcal{N}$ in a given feature of $\mathcal{N}$ as anchors, and choosing an arbitrary anchor if no corner for that layer appears in the feature. This gives an imperfect boxing for $\mathcal{N}$ if it has hidden layer-corners; we then make this imperfect boxing perfect by modifying some of these boxes, making thick or deep boxes to contain hidden layer-corners. The alignment properties ensure that the boxes that are thickened or deepened downward truncate the boxes below them, so that truncated boxes remain rectilinear.

Lemma 6 (Sufficiency) *If a niche $\mathcal{N}$ satisfies at least one of i)-v) in the Condition, then $\mathcal{N}$ is perfect.*

Proof. By Lemma 5, which establishes the perfection of any $\mathcal{N}$ with feature v), we can assume from now on that $\mathcal{N}$ has no keyless $\hat{3}$-corner. For each of the remaining features i)-iv), we provide a perfect boxing for $\mathcal{N}$ when $\mathcal{N}$ has that feature.

case i) Refer to Figures 6 and 7. Suppose $\mathcal{N}$ has a $V_{\hat{3}\hat{1}}$-layer containing cube $\hat{c}_{\mathcal{T}}$, which is a $V_{\mathcal{T}^\perp}$-layer or a $V_{\mathcal{S}^\perp}$-layer. Let $\hat{c}_3 = c_{\mathcal{T}}$ and $\hat{c}_1$ denote the extreme cubes at heights 3 and 1 in the $\hat{3}\hat{1}$-layer of $\hat{c}_3$. Let $\tilde{c}_2$ denote the cube just under $\hat{c}_3$. Cube $\tilde{c}_2$ is a layer-corner of layer 2, but not a corner of $\mathcal{N}$. Using $\hat{c}_3$, $\tilde{c}_2$, and $\hat{c}_1$ as anchors, create an ALB for each layer of $\mathcal{N}$. The boxes for $\hat{c}_3$, $\tilde{c}_2$, and $\hat{c}_1$ reach both the $\mathcal{S}$ and $\mathcal{T}$-walls, with

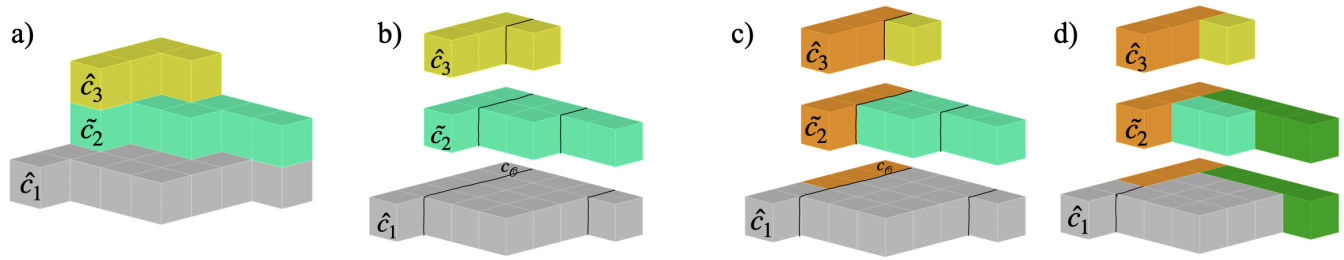


Figure 6: Sufficiency case (i) when a) the $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_3$ is a $V_{\mathcal{S}^\perp}$-layer. b) ALB's for the layers; c) box of $\hat{c}_3 = \hat{c}_\mathcal{T}$ extruded to $z = 0$ and to $x = 0$, making the deep orange box; d) box of $\hat{c}_\mathcal{S}$ extruded to $z = 0$ and to $y = 1$, making the thick green box.

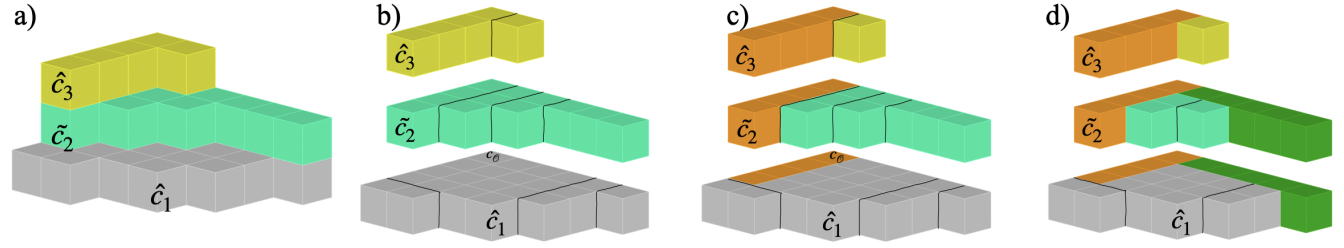


Figure 7: Sufficiency case (i) when a) the $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_3$ is a $V_{\mathcal{T}^\perp}$-layer. b) ALB's for the layers; c) box of $\hat{c}_\mathcal{T}$ extruded to $z = 0$ and to $x = 0$, making the deep orange box; d) box of $\hat{c}_\mathcal{S}$ extruded to $z = 0$ and to $y = 1$, making the thick green box.

$box(\tilde{c}_2)$ exactly below $box(\hat{c}_3)$. Then extrude $box(\hat{c}_3)$ down to the plane $z = 0$, making a deep box containing both $\hat{c}_3$ and $c_\mathcal{O}$. This deep box absorbs all the cubes of $box(\tilde{c}_2)$ but does not affect any other boxes in the anchored boxing of layer 2, which is anchored by $\tilde{c}_2$. The remaining boxes of this layer all extend to the $\mathcal{S}$-wall and lie beyond the plane $y = 1$. If the $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_3$ is a $V_{\mathcal{T}^\perp}$-layer, the deep box truncates by one unit the box anchored by $\hat{c}_1$ so that the box of $\hat{c}_1$ now abuts the side of the deep box on the plane $y = 1$; otherwise, if the $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_3$ is a $V_{\mathcal{S}^\perp}$-layer, the deep box truncates $box(\hat{c}_1)$ so that it abuts the deep box at the plane $x = x(\hat{c}_1)$.

To complete the construction of a perfect boxing for $\mathcal{N}$, make a thick box containing $\mathcal{S}$ and extrude this thick box over to the plane $y = 1$, where the thick box abuts the deep box. This extrusion truncates by one unit the boxes of the ALBs for layers 1 and 2 that lie beyond the plane $y = 1$; it does not affect any thin boxes in layer 3, and it does not affect the deep box or any other boxes of layer 1 that do not reach the $\mathcal{S}$-wall. The thin, possibly truncated, boxes of the ALBs together with the thick box for $\mathcal{S}$ and the deep $box(\hat{c}_3)$ make a perfect boxing for $\mathcal{N}$.

Examples of the two cases where the $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_3$ is a $V_{\mathcal{S}^\perp}$-layer and a $V_{\mathcal{T}^\perp}$-layer, respectively, are illustrated in Figures 6 and 7. In Figure 6a, the $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_3$ is the $\mathcal{T}$-wall. Figure 6b shows ALBs based on $\hat{c}_3$, $\tilde{c}_2$, and $\hat{c}_1$. Figure 6c illustrates the extrusion of $box(\hat{c}_3)$ (shown in color), which absorbs $box(\tilde{c}_2)$ and truncates $box(\hat{c}_1)$ at $x = x(\hat{c}_3)$. Figure 6d shows the final extrusion of the thick box of $\mathcal{S}$.

In Figure 7a, the $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_3$ is a $V_{\mathcal{T}^\perp}$-layer. Figure 7b shows ALBs based on $\hat{c}_3$, $\tilde{c}_2$, and $\hat{c}_1$. Figure 7c illustrates the downward extrusion of $box(\hat{c}_3)$ (shown in color), which absorbs $box(\tilde{c}_2)$ and truncates $box(\hat{c}_1)$ at $y = y(\hat{c}_3) = 1$. Figure 7d shows the final extrusion of the thick box of $\mathcal{S}$.

case ii) Refer to Figure 8. Suppose $\mathcal{N}$ has a $V_{\hat{2}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer. Let $\hat{c}_2$ and $\hat{c}_1$ denote the corners of layers 2 and 1 in the $V_{\hat{2}\hat{1}}$-layer that is that is a $V_{\mathcal{S}^\perp}$-layer, and let $\hat{c}_3$ denote an arbitrary corner of $\mathcal{N}$ in layer 3. Using $\hat{c}_3$, $\hat{c}_2$, and $\hat{c}_1$ as anchors, create an anchored boxing for each layer of $\mathcal{N}$. Now make a thick box containing the short tower $\mathcal{S}$, and extrude it to the plane $y = y(\hat{c}_2) = y(\hat{c}_1)$. This thick box truncates by 1 unit the boxes of the anchored boxings that lie in layer 1 or layer 2 on the $\mathcal{S}$ side of the plane $y = y(\hat{c}_2) = y(\hat{c}_1)$. These truncated boxes now end at the thick box containing $\mathcal{S}$ rather than at plane $x = 0$. The boxes of layer 3 are not affected. Similarly, create a thick box containing the two cubes of $\mathcal{T}$ and extrude this thick box to the plane $x = 0$. This truncates by one unit those boxes in layers 2 and 3 that abut the plane $y = 0$. Boxes in layer 1 are not affected. After these modifications, the

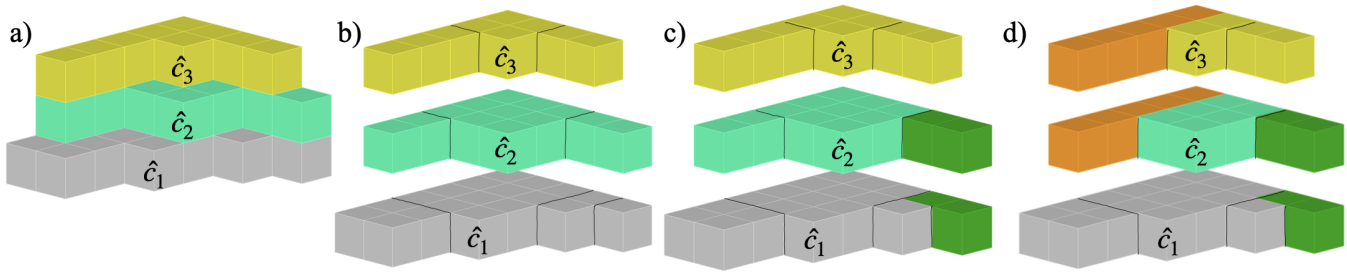


Figure 8: Sufficiency case (ii). a) $\mathcal{N}$ has a $V_{\hat{2}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer. b) ALB's for the layers; c) box of $\hat{c}_{\mathcal{S}}$ extruded to $z = 0$ and to $y = 2$, making the thick green box; d) box of $\hat{c}_{\mathcal{T}}$ extruded to $z = 1$ and to $x = 0$, making the thick orange box.

resulting set of boxes is a perfect boxing: each cube of $\mathcal{N}$ belongs to a box containing a corner.

case iii) Refer to Figure 9. Suppose $\mathcal{N}$ has a $V_{\hat{3}\hat{2}}$-layer that is $V_{\mathcal{T}^\perp}$-layer. This case is analagous to ii) above. Let $\hat{c}_3$ and $\hat{c}_2$ denote the corners of layers 3 and 2 in the $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer, and let $\hat{c}_1$ denote an arbitrary corner of $\mathcal{N}$ in layer 1. Using $\hat{c}_3$, $\hat{c}_2$, and $\hat{c}_1$ as anchors, create an anchored boxing for each layer of $\mathcal{N}$. Now make a thick box containing the tall tower $\mathcal{T}$, and extrude it to the plane $x = x(\hat{c}_3) = x(\hat{c}_2)$. This thick box truncates by 1 unit the boxes of the anchored boxings that lie in layer 2 or layer 3 on the $\mathcal{T}$ side plane $x = x(\hat{c}_3) = x(\hat{c}_2)$. These truncated boxes now end at the thick box containing $\mathcal{T}$ rather than at the $\mathcal{T}$-wall. The boxes of layer 1 are not affected. Similarly, create a thick box containing the short tower $\mathcal{S}$ and extrude this thick box to the $\mathcal{T}$-wall. This truncates by one unit those boxes in layers 1 and 2 that abut the $\mathcal{S}$-wall. Boxes in layer 3 are not affected. After these modifications, the resulting set of boxes is a perfect boxing: each cube of $\mathcal{N}$ belongs to a box containing a corner of $\mathcal{N}$.

case iv) Refer to Figure 10. Suppose $\mathcal{N}$ contains a $\hat{3}$-corner $\hat{c}_3$ whose V-layers together contain a $\hat{2}$-corner $\hat{c}_2$ and a $\hat{1}$-corner $\hat{c}_1$. If $\hat{c}_2$ and $\hat{c}_1$ belong to the same V-layer of $\hat{c}_3$, then either case ii) or case iii) applies. Likewise, case ii) or case iii) applies if the V-layers are distinct, and the $\hat{3}\hat{2}$-layer is a $V_{\mathcal{T}^\perp}$-layer. Thus we need only consider the situation in which the $\hat{3}\hat{2}$-layer is a $V_{\mathcal{S}^\perp}$-layer and the $\hat{3}\hat{1}$-layer is a $V_{\mathcal{T}^\perp}$-layer.

Using anchor cubes $\hat{c}_3$, $\hat{c}_2$, and $\hat{c}_1$, create an anchored boxing for each layer of $\mathcal{N}$. See Figure 10a and 10b. Now extrude $box^*(\hat{c}_3)$ to plane $z = 0$, creating a deep box for corner $\hat{c}_3$. See Figure 10c.

This deep box truncates $box^*(\hat{c}_2)$, which becomes a smaller box that measures $|x(\hat{c}_2) - x(\hat{c}_3)|$ in the x-dimension. This smaller box, denoted $box^\dagger(\hat{c}_2)$, has one face in plane $y = 0$ and abuts the deep box, on the plane $x = x(\hat{c}_3)$. Because $x(\hat{c}_3) < x(\hat{c}_2)$ and $y(\hat{c}_3) = y(\hat{c}_2)$, $box^*(\hat{c}_2)$ is the only box of the ALB for layer 2 affected by the deep box.

Similarly, the deep box also truncates $box^*(\hat{c}_1)$, which becomes a box that measures $|y(\hat{c}_1) - y(\hat{c}_3)|$ in the y-dimension. This smaller box, denoted $box^\dagger(\hat{c}_1)$, has a face on the plane $x = 0$ and abuts the deep box on the plane $y = y(\hat{c}_3)$. Because $y(\hat{c}_1) > y(\hat{c}_3)$ and $x(\hat{c}_3) = x(\hat{c}_1)$, $box^*(\hat{c}_1)$ is the only box of the anchored boxing of layer 1 that is affected by the deep box.

Now box together the two cubes of $\mathcal{S}$, making a thick box for corner $\hat{c}_{\mathcal{S}}$. Extrude this thick box to the plane $y = y(\hat{c}_3)$, so that the thick box abuts the deep box. See Figure 10d. Also, box together the two cubes of $\mathcal{T}$, making a thick box for corner $\hat{c}_{\mathcal{T}}$, and extrude this thick box to the plane $x = x(\hat{c}_3)$, so that it abuts the deep box. See Figure 10e. The extrusion of the thick box containing $\mathcal{S}$ truncates by one unit any boxes of the anchored boxings for layers 1 and 2 (including $box^\dagger(\hat{c}_1)$) that lie outside the deep box and that reach the plane $x = 0$. The extrusion does not affect any boxes of layer 3. The extrusion of the thick box containing the cubes of $\mathcal{T}$ truncates by one unit any boxes of the anchored boxings for layers 3 and 2 (including $box^\dagger(\hat{c}_2)$) that lie outside the deep box and that reach the $y = 0$ plane. The extrusion does not affect any boxes of layer 1.

The resulting boxing is a perfect boxing for $\mathcal{N}$: each cube belongs to the box of a corner of $\mathcal{N}$, and the interiors of the boxes are pairwise disjoint.

This completes the constructions for cases i)-v) and concludes the proof. □

4 Necessity: Alignments in Perfect Niches

Technical Lemmas 7–12 show how i)-v) of the Condition can arise from a perfect boxing $\mathcal{B}$ of a niche $\mathcal{N}$. Each element of a perfect $\mathcal{B}$ is a box b, denoted $b = box(c)$, where c is any cube inside the box, and the box must contain a corner. A box is called an $\hat{i}$-*box*, $1 \le i \le 3$, if it contains an $\hat{i}$-corner.

The proof of Lemma 13 pulls Lemmas 7–12 together and establishes the necessity of the Condition. The proof, given a perfect boxing $\mathcal{B}$, uses the $box(c_{\mathcal{O}})$-*method*, which considers which $\hat{i}$-box contains $c_{\mathcal{O}}$.

We introduce our $box(c_{\mathcal{O}})$-method by using it to prove

that the triskele is not perfect. In the proof, we say a cube c is *dominated* by a corner if c lies in the box determined by the corner and $c_{\mathcal{O}}$. Cubes that are dominated by only one corner must lie in the box of that corner in any perfect boxing.

The $box(c_{\mathcal{O}})$-method, used to prove triskeles are not perfect. A triskele has only three corners, $\hat{c}_1$, $\hat{c}_2$, $\hat{c}_3$, one in each layer i, $1 \leq i \leq 3$. Each $\hat{c}_i$ has two more cubes that must belong to $box(\hat{c}_i)$ as those cubes are uniquely dominated. Since each $box(\hat{c}_i)$ is convex, it contains a fourth cube (see Figure 3b and Figure 3c). If $c_{\mathcal{O}}$ belongs to $box(\hat{c}_i)$, that box contains six cubes. Thus the triskele must have at least 14 cubes, a contradiction. Hence the triskele is not perfect. Optimum boxings $\mathcal{B}$ with $\|\mathcal{B}\|$=4 are shown in Figure 3b, where $c_{\mathcal{O}}$ is boxed by itself, and Figure 3c, where $box(c_{\mathcal{O}})$ is deep.

We give some notation. Cube $c_{x',y',z'}$ is a lower *neighbor*, or simply a *neighbor*, of corner $\hat{c}_{x,y,z}$ if $x' = x$, $y' = y+1$, and $z' < z$, and similarly if $x' = x+1$, $y' = y$, and $z' < z$. The $\hat{3}$-corner $\hat{c}_3$ of the triskele in Figure 3 has two neighbors in layer 1 and no neighbors in layer 2.

For each of Lemmas 7–12, we assume that niche $\mathcal{N}$ has a perfect boxing $\mathcal{B}$.

Lemma 7 *If $box(\hat{c}_{\mathcal{T}}) \in \mathcal{B}$ is deep, $\mathcal{N}$ has feature i).*

Proof. By definition of tower $\mathcal{T}$, the key cube c_{key} of $\hat{c}_{x_{\mathcal{T}},1,3} = \hat{c}_{\mathcal{T}}$ exists. At least one of the layer 1 neighbors of $\hat{c}_{\mathcal{T}}$ is not boxed with c_{key} , as otherwise, $box(c_{key})$ would intersect $box(\hat{c}_{\mathcal{T}})$. If $c_{key} \notin box(c_{x_{\mathcal{T}}+1,1,1})$ then the corner of this box lies in the $\mathcal{T}$-wall, which is thus a $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_{\mathcal{T}}$. If $c_{key} \notin box(c_{x_{\mathcal{T}},2,1})$ then the corner of this box lies in the $V_{\mathcal{T}^\perp}$-layer of $\hat{c}_{\mathcal{T}}$. □

Lemma 8 *If $\mathcal{B}$ has a $\hat{2}$-box that contains a cube of the $\mathcal{T}$-wall, then $\mathcal{N}$ has feature iii).*

Proof. Let x' be the maximum x-coordinate of a cube in the $\mathcal{T}$-wall that belongs to a $\hat{2}$-box. Since $x' < x_{\mathcal{T}}$, cube $c_{x',1,3}$ exists and its box is thin. We claim that the $\hat{3}$-corner of $box(c_{x',1,3})$ lies in the same $V_{\mathcal{T}^\perp}$-layer that contains the $\hat{2}$-corner of $box(c_{x',2,1})$, by the maximality of x': otherwise, the x-coordinate x'' of the $\hat{3}$-corner of $box(c_{x',1,3})$ would be greater than x', and would contain a cube $c_{x'',1,3}$ in the $\mathcal{T}$-wall, contradicting the maximality of x'. This $V_{\mathcal{T}^\perp}$ V-layer is thus a $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer so $\mathcal{N}$ has feature iii). □

The niche in Figure 5 has one regular $\hat{3}$-corner, $\hat{c}_3$. This regular $\hat{3}$-corner has four neighbors, two in each of its V-layers. The layer 2 neighbor $\hat{c}'_2$ in its $V_{\mathcal{T}^\perp}$-layer is a $\hat{2}$-corner, as is the layer 2 neighbor $\hat{c}_2$ in its $V_{\mathcal{S}^\perp}$ layer.

The next definition is used for the analysis of deep boxes and the corner alignments that arise from them.

Definition [*regular $\hat{3}$-corner*]. A $\hat{3}$-corner $\hat{c}_{x,y,3}$ is *regular* if: i) it has a key cube, and ii) $\hat{c}_{x,y,3} \neq \hat{c}_{\mathcal{T}}$.

The niche in Figure 5 has one regular $\hat{3}$-corner, $\hat{c}_3$. This regular $\hat{3}$-corner has four neighbors, two in each of its V-layers. The layer 2 neighbor $\hat{c}_2$ in its $V_{\mathcal{T}^\perp}$-layer is a $\hat{2}$-corner, as is the layer 2 neighbor $\hat{c}'_2$ in its $V_{\mathcal{S}^\perp}$-layer.

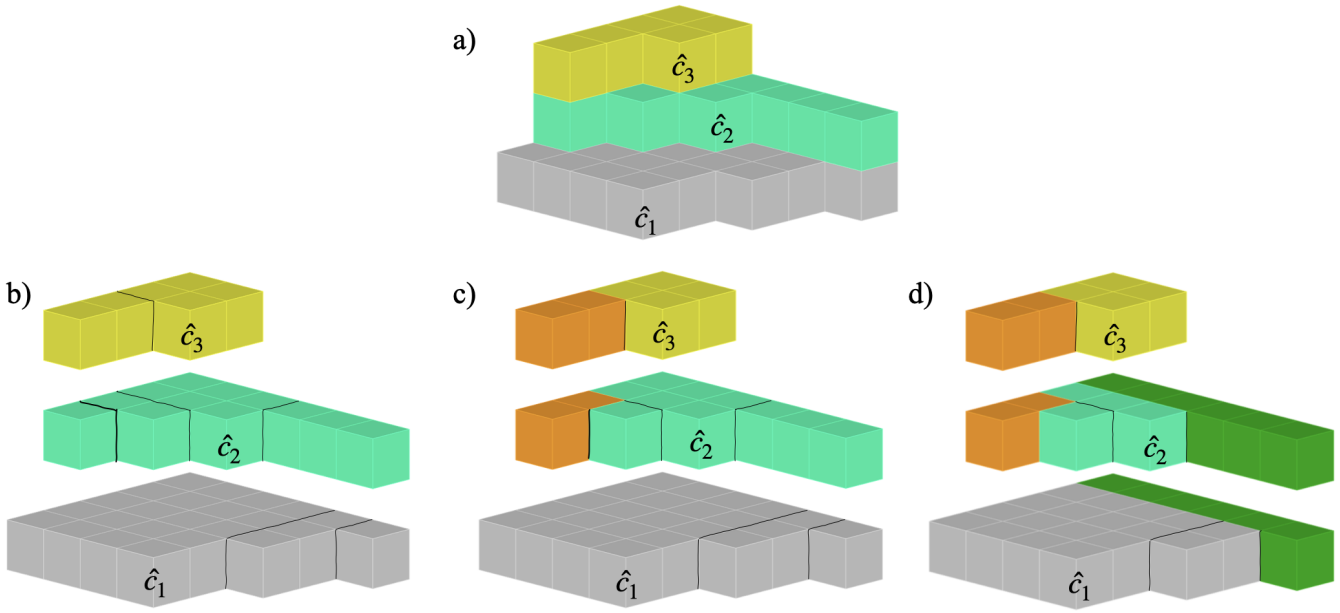


Figure 9: Sufficiency case (iii). a) $\mathcal{N}$ has a $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer. b) ALB's for the layers; c) box of $\hat{c}_{\mathcal{T}}$ extruded to $z-1$ and to $x=2$, making the thick orange box; d) box of $\hat{c}_{\mathcal{S}}$ extruded to $z=0$ and to $y=0$, making the thick green box.

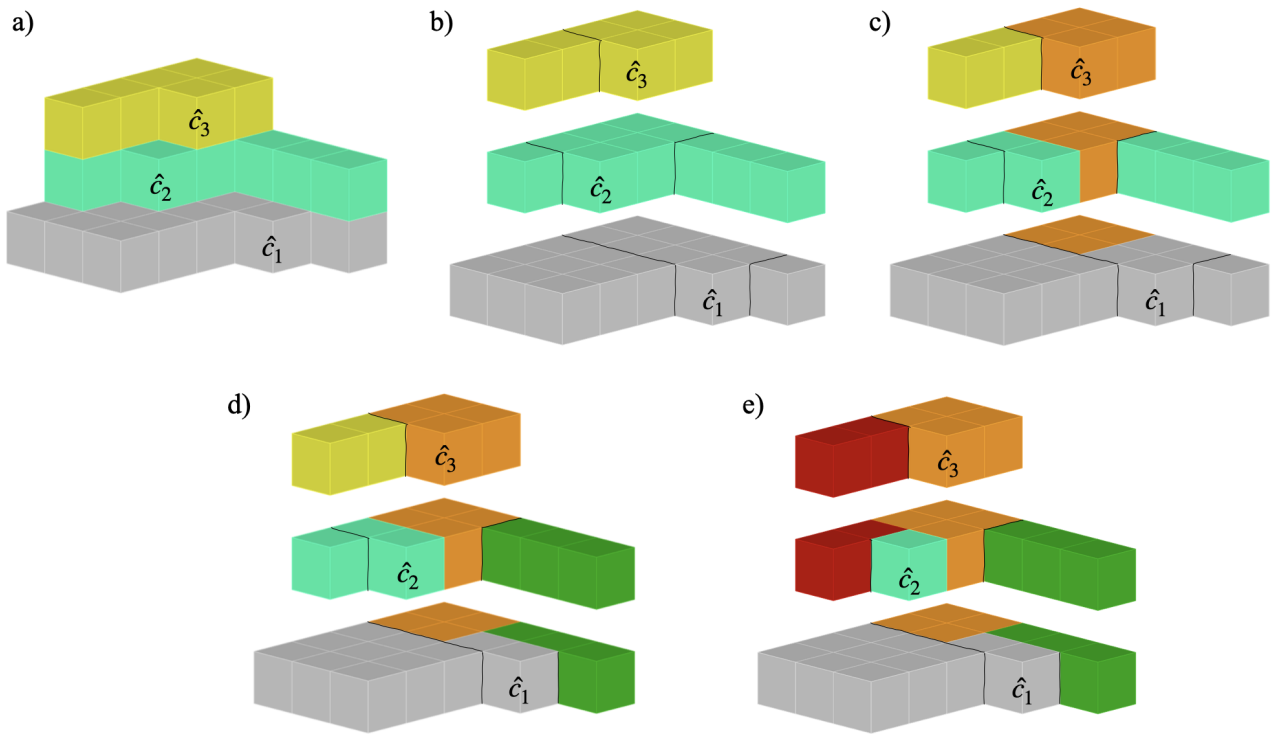


Figure 10: Sufficiency case (iv). a) $\mathcal{N}$ has a $\hat{3}$-corner $\hat{c}_3$ whose V-layers together have a $\hat{2}$-corner $\hat{c}_2$ and a $\hat{1}$-corner $\hat{c}_1$. b) ALB's for each layer; c) box of $\hat{c}_3$ extruded to $z = 0$, creating the deep orange box; d) box of $\hat{c}_{\mathcal{S}}$ extruded to $z = 0$ and to $y = 2$, creating the thick green box; e) box of $\hat{c}_{\mathcal{T}}$ extruded to $z = 1$ and $x = 2$, creating the thick red box.

Lemma 9 *If $\hat{c}_{x,y,3}$ is regular and $box(\hat{c}_{x,y,3}) \in \mathcal{B}$ is thick or deep, then $\hat{c}_{x,y,3}$ has a $V_{\hat{3}\hat{2}}$-layer.*

Proof. Since cube $key(\hat{c}_{x,y,3})$ exists, the layer 1 neighbors of $\hat{c}_{x,y,3}$ also exist. Since $\hat{c}_{x,y,3} \neq \hat{c}_{\mathcal{T}}$, and since there are no other tall towers in $\mathcal{N}$, corner $\hat{c}_{x,y,3}$ has at least one layer 2 neighbor. If there is a cube in layer 2 over the key, then the regular $\hat{3}$-corner has two neighbors in layer 2. They cannot belong to the same box, as such a box would intersect the thick or deep $box(\hat{c}_{x,y,3})$. A neighbor of layer 2 that is not boxed with the cube above the key is boxed with a $\hat{2}$-corner in a V-layer of $\hat{c}_{x,y,3}$, which is a $V_{\hat{3}\hat{2}}$-layer. Likewise, if there is no cube in layer 2 over the key, then any layer 2 neighbor of $\hat{c}_{x,y,3}$ is boxed with a $\hat{2}$-corner in a $V_{\hat{3}\hat{2}}$-layer of $\hat{c}_{x,y,3}$. □

The proof of the next lemma is similar.

Lemma 10 *If $\hat{c}_{x,y,3}$ is regular and $box(\hat{c}_{x,y,3}) \in \mathcal{B}$ is deep, then one of the neighbors of $\hat{c}_{x,y,3}$ in layer 1 belongs to a box whose corner lies in a V-layer of $\hat{c}_{x,y,3}$, where this V-layer is a $V_{\hat{3}\hat{2}}$-layer or a $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_{x,y,3}$.*

Proof. The key cube exists, so $\hat{c}_{x,y,3}$ has two neighbors in layer 1. They cannot belong to the same box, as this would intersect $box(\hat{c}_{x,y,3})$, so at least one neighbor in layer 1 is not boxed with the key cube; hence, the corner of the box of this neighbor, where the box can be thick or thin, belongs to a V-layer of $\hat{c}_{x,y,3}$. This V-layer is thus a $V_{\hat{3}\hat{2}}$-layer or a $V_{\hat{3}\hat{1}}$-layer of $\hat{c}_{x,y,3}$. □

Lemma 11 *If $\hat{c}_{x_0,y_0,3}$ is a regular $\hat{3}$-corner with a $V_{\hat{3}\hat{1}}$-layer, and if $box(\hat{c}_{x_0,y_0,3}) \in \mathcal{B}$ is deep, then $\mathcal{N}$ has feature ii), iii), or iv) of the Condition.*

Proof. By Lemma 9, the regular corner $\hat{c}_{x_0,y_0,3}$ has a V-layer containing a $\hat{2}$-corner. If this V-layer contains a $\hat{1}$-corner, then the V-layer is a $V_{\hat{3}\hat{2}\hat{1}}$-layer. If this $V_{\hat{3}\hat{2}\hat{1}}$-layer is a $V_{\mathcal{T}^\perp}$-layer, then $\mathcal{N}$ has feature iii) of the Condition. If the layer is a $V_{\mathcal{S}^\perp}$-layer, then $\mathcal{N}$ has feature ii). If the regular corner $\hat{c}_{x_0,y_0,3}$ has a $\hat{2}$-corner in one V-layer and a $\hat{1}$-corner in the other V-layer, then $\mathcal{N}$ has feature iv). □

Definition [*special box*]. A $box(\hat{c}_{x_0,y_0,3})$ is *special* if: i) the box is deep, and ii) $\hat{c}_{x_0,y_0,3}$ is a regular $\hat{3}$-corner with no $V_{\hat{3}\hat{1}}$-layer and no $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer.

Lemma 12 *If $\mathcal{B}$ contains a special box, then $\mathcal{N}$ satisfies the Condition.*

Proof. The box is deep and $\hat{c}_{x_0,y_0,3}$ is regular but has no $V_{\hat{3}\hat{1}}$-layer and no $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer. It follows from Lemma 10 that the $V_{\mathcal{S}^\perp}$-layer of $\hat{c}_{x_0,y_0,3}$

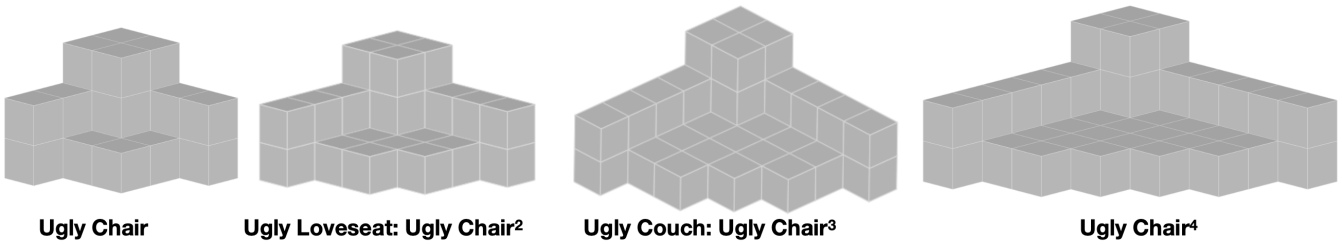


Figure 11: A collection of *ugly furniture*: each cornerhedron is imperfect, consists of two niches sharing a tall tower, and has an optimal boxing of size $k+1$.

has a $\hat{2}$-corner and that the box of the $\hat{2}$-corner must be thick as this box contains $c_{x_0+1,y_0,1}$. Hence the $\hat{2}$-corner box contains both $c_{x_0+1,y_0,1}$ and $c_{x_0+1,y_0,2}$.

Consider all the corners of thick $\hat{2}$-boxes (we have just seen that there is at least one), and let $\hat{c}_{x'_0,y'_0,2}$ denote the corner with maximum x-coordinate x'_0. Hence $box(\hat{c}_{x'_0,y'_0,2})$ is a thick $\hat{2}$-box, and with respect to the position of $\hat{c}_{x_0,y_0,3}$, the coordinates of $\hat{c}_{x'_0,y'_0,2}$ satisfy $x'_0 > x_0$ and $y'_0 \leq y_0$. If $box(\hat{c}_{x'_0,y'_0,2})$ contains cubes in the $\mathcal{T}$-wall, then by Lemma 8, $\mathcal{N}$ satisfies the Condition with feature iii). If $box(\hat{c}_{x'_0,y'_0,2})$ does not reach the $\mathcal{T}$ wall, then we continue searching in $\mathcal{N}$ for a feature of the Condition. Let $c_{x'_0,y''_0,1}$ and $c_{x'_0,y''_0,2}$, where $y''_0 < y'_0$, denote the two cubes in the $V_{\mathcal{T}^\perp}$-layer of $\hat{c}_{x'_0,y'_0,2}$ that abut $box(\hat{c}_{x'_0,y'_0,2})$. We consider two cases i) and ii).

Case i) y''_0=1: The two cubes lie in the $\mathcal{T}$-wall and $box(c_{x'_0,y''_0,2}) = box(\hat{c}_{x_T,1,3}) = box(\hat{c}_{\mathcal{T}})$. If $box(\hat{c}_{\mathcal{T}})$ contains $c_{x'_0,y''_0=1,1}$ and so is deep, then by Lemma 7, $\mathcal{N}$ has feature i) of the Condition; otherwise, $box(c_{x'_0,y''_0=1,1})$ belongs to the $\mathcal{T}$-wall and contains a $\hat{1}$-corner. Again the $\mathcal{T}$-wall is a $V_{\hat{3}\hat{1}}$-layer and $\mathcal{N}$ has feature i).

Case ii) $y''_0 > 1$: The corners of $box(c_{x'_0,y''_0,1})$ and $box(c_{x'_0,y''_0,2})$, which may or not be the same box, lie in the V-layer that contains the two cubes $c_{x'_0,y''_0,1}$ and $c_{x'_0,y''_0,2}$, where this V-layer is parallel to, but distinct from, the $\mathcal{T}$-wall; otherwise, their box(es) would intersect the thick $box(\hat{c}_{x'_0,y'_0,2})$. We denote this V-layer by V'' (its cubes have the same y-coordinate y''_0).

Suppose $c_{x'_0,y''_0,1}$ and $c_{x'_0,y''_0,2}$ belong to the same box B''. By maximality of x'_0, box B'' cannot be a thick $\hat{2}$-box, so B'' must be a deep $\hat{3}$-box. However, the $\hat{3}$-corner of B'' cannot be $\hat{c}_{\mathcal{T}}$, as B'' contains cubes in V''. Moreover, B'' cannot be special, by maximality of x_0. Hence either its $\hat{3}$-corner is not regular because it has no key, and thus $\mathcal{N}$ has feature v), or the $\hat{3}$-corner of B is regular. In this case, since B is not special, the $\hat{3}$-corner either lies in a $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer, and thus $\mathcal{N}$ has feature iii), or the $\hat{3}$-corner lies in a $V_{\hat{3}\hat{1}}$-layer, and thus $\mathcal{N}$ has feature ii), iii) or iv) by Lemma 11.

To complete the proof of case ii) and the proof of the lemma, suppose $c_{x'_0,y''_0,1}$ and $c_{x'_0,y''_0,2}$ do not belong to the same box. Then $box(c_{x'_0,y''_0,1})$ is a $\hat{1}$-box with $\hat{1}$-corner in V'', and $box(c_{x'_0,y''_0,2})$ is either a $\hat{2}$-box or a $\hat{3}$-box with corner in V''. If $box(c_{x'_0,y''_0,2})$ is a $\hat{2}$-box, then V'' is a $V_{\hat{2}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer, so $\mathcal{N}$ has feature ii). If $box(c_{x'_0,y''_0,2})$ is a $\hat{3}$-box with corner $\hat{c}_{x''_0,y''_0,3}$ then this box is thick (not deep, as it lies above $box(c_{x'_0,y''_0,1})$). If $\hat{c}_{x'',y'',3}$ has no key, then $\mathcal{N}$ has feature v); otherwise, the key exists and by Lemma 9, at least one V-layer of $\hat{c}_{x''_0,y''_0,3}$ is a $V_{\hat{3}\hat{2}}$-layer. If this V-layer is V'', then V'' is a $V_{\hat{3}\hat{2}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer and $\mathcal{N}$ has feature ii). Otherwise, $\hat{c}_{x''_0,y''_0,3}$ has a $\hat{2}$-corner in one of its V-layers (namely in its $V_{\mathcal{T}^\perp}$-layer) and a $\hat{1}$-corner in the other (namely in layer V'', its $V_{\mathcal{S}^\perp}$-layer), so $\mathcal{N}$ has feature iv). □

We now establish the necessity of the Condition with the $box(c_{\mathcal{O}})$-method. The proof considers perfect boxings $\mathcal{B}_i$ such that $box(\mathcal{O})$ is an $\hat{i}$-box, $1 \leq i \leq 3$, and shows that in each case, $\mathcal{N}$ has at least one of features i)-v).

Lemma 13 *The Condition of Theorem 3 is necessary.*

Proof. $\hat{i} = 1$) Some perfect boxing $\mathcal{B}_1$ boxes the origin cube $c_{\mathcal{O}}$ in a thin $\hat{1}$-box: Let $c_{1,y',1}$ denote the cube in the $\mathcal{S}$-wall with maximum y-coordinate y' such that the cube belongs to a thin $\hat{1}$-box, namely $box(\hat{c}_{x',y',1})$, which may or not be equal to $box(c_{\mathcal{O}})$. Since $y' < y_{\mathcal{S}}$, there must exist a cube $c_{1,y',2}$ above $c_{1,y',1}$. By maximality of y', the corner of $box(c_{1,y',2})$, which may be a $\hat{2}$- or $\hat{3}$-corner, lies in the $V_{\mathcal{S}^\perp}$-layer containing $\hat{c}_{x',y',1}$, so this V-layer is either a $V_{\hat{2}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer, or a $V_{\hat{3}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer. In the case of a $V_{\hat{2}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer, $\mathcal{N}$ has feature ii). The case of a $V_{\hat{3}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer arises when $box(c_{1,y',2})$ has a $\hat{3}$-corner and hence is thick. If the $\hat{3}$-corner is regular, then by Lemma 9 the $\hat{3}$-corner has a $V_{\hat{3}\hat{2}}$-layer, and also, its $V_{\mathcal{S}^\perp}$-layer is a $V_{\hat{3}\hat{1}}$-layer. If the two V-layers are the same, the $\hat{3}$-corner has a $V_{\hat{3}\hat{2}\hat{1}}$-layer that is a $V_{\mathcal{S}^\perp}$-layer and $\mathcal{N}$ has feature ii); if the layers are distinct, then $\mathcal{N}$ has feature iv). To complete this case, if the $\hat{3}$-corner of $box(c_{1,y',2})$ is not regular, either the $\hat{3}$-corner is keyless, or it belongs to $\mathcal{T}$. If the $\hat{3}$-corner is keyless, then $\mathcal{N}$ has feature v). If the $\hat{3}$-corner is equal to $\hat{c}_{\mathcal{T}}$, then its $V_{\mathcal{S}^\perp}$-layer is the $\mathcal{T}$-wall, which is a $V_{\hat{3}\hat{1}}$-layer, so $\mathcal{N}$ has feature i).

$\hat{i} = 2$) Some perfect boxing $\mathcal{B}_2$ boxes the origin cube $c_{\mathcal{O}}$ in a thick $\hat{2}$-box: By Lemma 8, $\mathcal{N}$ has feature iii).

$\hat{i} = 3$) Some perfect boxing $\mathcal{B}_3$ boxes $c_{\mathcal{O}}$ in a deep $\hat{3}$-box: If the $\hat{3}$-corner of this deep box is $\hat{c}_{\mathcal{T}}$, then by Lemma 7, $\mathcal{N}$ has feature i). If the $\hat{3}$-corner of the deep box is keyless, then $\mathcal{N}$ has feature v).

If the $\hat{3}$-corner of the deep box is not $\hat{c}_{\mathcal{T}}$ and the $\hat{3}$-corner is keyed, then by definition this corner is regular. If this regular $\hat{3}$-corner of deep $box(c_{\mathcal{O}})$ has a V-layer containing a $\hat{1}$-corner, then by Lemma 11, $\mathcal{N}$ has feature ii), iii), or iv). If the $\hat{3}$-corner does not have such a V-layer, then either the $\hat{3}$-corner has a $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer, or the box of this $\hat{3}$-corner is a special deep box. If the $\hat{3}$-corner has a $V_{\hat{3}\hat{2}}$-layer that is a $V_{\mathcal{T}^\perp}$-layer then $\mathcal{N}$ has feature iii). If the box of the $\hat{3}$-corner is special, then by Lemma 12, $\mathcal{N}$ has at least one of features i) - v).

Thus any perfect boxing $\mathcal{B}$ of $\mathcal{N}$ has at least one of features i)-v). □

5 Discussion and Conclusion

We have characterized perfect niches and constructed optimum boxings for all niches, i.e., boxings $\mathcal{B}$ of size $\|\mathcal{B}\| = k$ for perfect niches and of size $\|\mathcal{B}\| = k+1$ for imperfect niches. To do this, we developed two methods: the anchored layer-boxing (ALB) method for constructing partitions, and the $box(c_{\mathcal{O}})$ method for case analysis of a given perfect boxing.

Our work has focussed on characterization of perfect niches. Our theorems and their proofs clearly suggest how to achieve low running time algorithms for recognizing perfect niches and for constructing an optimum boxing of any given niche. Depending on the details of the input and data structures chosen, we conjecture that algorithms for solving these two problems have running times of $O(k \lg k)$, where k is the number of corners.

Many open problems arise for 3-layered cornerhedra and more generally, for cornerhedra of three or more layers.

1) Based on our characterization of perfect niches, provide an efficient algorithm and implementation for recognizing perfect niches, representing them succinctly, e.g., by the coordinates of their corners.

2) Provide an efficient algorithm for boxing niches optimally, representing them succinctly.

3) Determine the complexity of recognizing perfect 3-layer cornerhedra.

4) Determine the complexity of boxing 3-layer cornerhedra optimally.

5) Given a subfamily (i.e., class) of cornerhedra, how large can the gap be between the size $\|\mathcal{B}(\mathcal{C})\|$ of an optimum boxing for a member $\mathcal{C}$ of the class and the number of corners $k(\mathcal{C})$ of $\mathcal{C}$? For example, for 3-layer cornerhedra, we can obtain a lower bound for the gap by cascading triskeles as in Figure 12. As another example, we define a subfamily of 3-layer cornerhedra we call *ugly furniture* to consist of two niches that share a tall tower. Optimum boxings of the subset of ugly furniture shown in Figure 11 are of size $k + 1$, where k is the number of corners of the niche, so the gap is at least 1. Is it exactly 1?

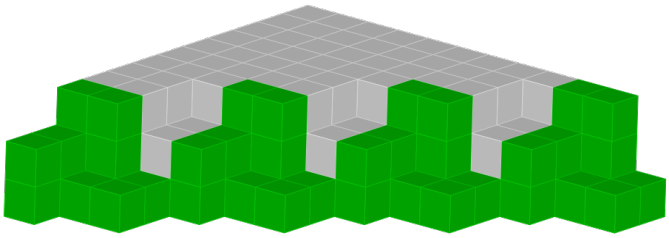


Figure 12: Cascadia: $opt\|\mathcal{B}\| - k = 4$

6) Enumerate perfect niches for given bounding box sizes, and likewise for imperfect niches. Counting perfect niches (or imperfect niches, or niches) mirrors a problem of MacMahon, which, in his terminology, was to count the number of (a, b, c)-plane partitions. In our terminology, this is equivalent to counting the number of cornerhedra of height at most c in an $a \times b \times c$ bounding box. MacMahon [12] gave a well-known formula for this.

Acknowledgements

This research was initiated at two International Workshops on Computational Geometry and its Applications to Computer Graphics, co-organized by Sylvain Lazard and Sue Whitesides at the Bellairs Research Institute of McGill; partial research funding was provided by NSERC DGs of Stege and Whitesides. We thank Noah Janssen, Harvey Ratson, and the workshop participants for discussions and continued interest. We also thank the reviewers for their very helpful comments.

References

[1] P.K. Agarwal, M. Sharir, A. Steiger. Decomposing the complement of the union of cubes in three dimensions. In Proc. of *SODA 2021*, ACM-SIAM Symposium on Discrete Algorithms, pp. 1425-1444

[2] T.C. Biedl, M. Derka, V. Irvine, A. Lubiw, D. Mondal, and A. Turcotte. Partitioning orthogonal histograms into rectangular boxes. In Proc. of *LATIN 2018*, Lecture Notes in Computer Science, vol. 10807, pp. 146–160. Springer, 2018.

[3] V. Chekanin. Solving the Problem of Packing Objects of Complex Geometric Shape into a Container of Arbitrary Dimension. CEUR Workshop Proceedings, 2744 (2020). doi:10.51130/graphicon-2020-2-3-50.

[4] V.J. Dielissen and A. Kaldewaij. Rectangular partition in polynomial in two dimensions but NP-complete in three. *Inf. Process. Lett.*, 38:1–6, 1991.

[5] D. Eppstein and E. Mumford. Steinitz theorems for simple orthogonal polyhedra. *J. Comput. Geom.*, 5(1):179–244, 2014.

[6] L. Ferrari, P.V. Sankar, and J. Sklansky. Minimal rectangular partitions of digitized blobs. *Computer Vision, Graphics, and Image Processing*, 28(1):58–71, 1984.

[7] P. Floderus, J. Jansson, C. Levcopoulos, A. Lingas, D. Sledneu. 3D rectangulations and geometric matrix multiplication. *Algorithmica*, 80:136-54, 2018.

[8] R. Gomory. Origin and early evolution of corner polyhedra. *European Journal of Operational Research*, 253: 543–55, 2016.

[9] C. Hoschl and J. Flusser. Decomposition of 3D Binary Objects into Rectangular Blocks. In Proc. *International Conference on Digital Image Computing: Techniques and Applications (DICTA2016)*, pp. 1-8, 2016.

[10] W. Lipski Jr., E. Lodi, F. Luccio, C. Mugnai, and L. Pagli. On two-dimensional data organization II. *Fundamenta Informaticae*, 2:245–260, 1979.

[11] M. Livesu, S. Ellero, J. Martínez, S. Lefebvre and M. Attene. From 3D models to 3D prints: an overview of the processing pipeline. *Comput. Graph. Forum*, 36(2): 537–64, 2017.

[12] P.A. MacMahon. Combinatory Analysis vols. 1, 2. Cambridge University Press, 1915, 1916; reprinted by Chelsea, New York (1960), and by Dover, New York (2004)

[13] T. Ohtsuki. Minimum dissection of rectilinear regions. In Proc. *IEEE Int. Symp. Circuits and Systems*, pp. 1210–1213, 1982.

[14] P. Winkler. Mathematical Puzzles: A Connoisseur's Collection. CRC Press, 2003.

The Exact Routing and Spanning Ratio of arbitrary triangle Delaunay graphs

Prosenjit Bose * Jean-Lou De Carufel † John Stuart ‡

Abstract

A Delaunay graph built on a planar point set has an edge between two vertices when there exists a disk with the two vertices on its boundary and no vertices in its interior. When the disk is replaced with an equilateral triangle, the resulting graph is known as a *Triangle-Distance Delaunay Graph* or TD-Delaunay for short. A generalized $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph is a TD-Delaunay graph whose empty region is a scaled translate of a triangle with angles of $\theta_1, \theta_2, \theta_3 := \pi - \theta_1 - \theta_2$ with $\theta_1 \leq \theta_2 \leq \theta_3$. We prove that $\frac{1}{\sin(\theta_1/2)}$ is a lower bound on the spanning ratio of these graphs which matches the best known upper bound (Lubiw & Mondal *J. Graph Algorithms Appl.*, **23**(2):345–369). Then we provide an online local routing algorithm for $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs with a routing ratio that is optimal in the worst case. When $\theta_1 = \theta_2 = \frac{\pi}{3}$, our expressions for the spanning ratio and routing ratio evaluate to 2 and $\frac{\sqrt{5}}{3}$, matching the known tight bounds for TD-Delaunay graphs.

1 Introduction

Geometric graphs are graphs whose vertex sets are points in the plane and whose edge weights are the corresponding Euclidean distances. A common theme in Computational Geometry is the study of shortest paths. In geometric graphs, one measure of how well a graph preserves distances is its spanning ratio. The spanning ratio of a geometric graph is the smallest upper bound on the ratio of distance in the graph to distance in the plane for all pairs of points [11]. One particular geometric graph of interest is the Delaunay triangulation, which has an edge between two vertices exactly when they lie on the boundary of a disk which contains no other vertex in its interior [9].

A long-standing open problem is to determine the worst-case spanning ratio of the Delaunay triangulation, which is known to be between 1.5932 [16] and 1.998 [15]. In other words, there exists a point set where the spanning ratio is at least 1.5932, and for any point set, the spanning ratio is at most 1.998. While the exact spanning ratio of the standard Delaunay triangulation remains unknown, several variants do have known tight spanning ratios in the worst case. For example, when the empty disk is replaced with a square we obtain the L_∞ or L_1-Delaunay graph, which is known to have a spanning ratio of exactly $\sqrt{4+2\sqrt{2}} \approx 2.61$ [3]. Similar proof techniques have been generalized to Delaunay graphs based on rectangles and parallelograms [14, 12]. In general, one can define a Delaunay graph from any convex distance function, and such a graph is known to have a constant spanning ratio where the spanning ratio depends on the ratio of the perimeter to the width of the convex shape [4]. When the unit circle in this distance is a regular hexagon, then the exact worst-case spanning ratio is 2 [13]. When the unit circle is an equilateral triangle, then exact worst-case spanning ratio is also 2 [8]. A generalized $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph is a TD-Delaunay graph whose empty region is a scaled translate of a triangle with angles of $\theta_1, \theta_2, \theta_3 := \pi - \theta_1 - \theta_2$ with $\theta_1 \leq \theta_2 \leq \theta_3$. In this paper, we provide a lower bound of $\frac{1}{\sin(\theta_1/2)}$ that matches the best known upper bound for $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs [10].

The routing ratio of a geometric graph essentially captures how feasible it is to find short paths in a graph when making local decisions based only on the neighbourhood of the current vertex. The routing ratio is the smallest upper bound on the ratio of the length of the path returned by the routing algorithm and the Euclidean distance between all pairs of vertices. Routing in Delaunay trianglulations is notoriously difficult, with the routing ratio of the standard Delaunay triangulation known to be between 1.70 [1] and 3.56 [1]. Variations such as the L_1-Delaunay triangulation are known to have a routing ratio between 2.7 [1] and 3.16 [7].

For TD-Delaunay graphs, there is a gap between the spanning ratio of 2 and the routing ratio which was shown to be exactly $\frac{5}{\sqrt{3}}$ in the worst-case [5]. We show that this gap is preserved for $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs by extending techniques from [5] to obtain a tight routing ratio of

$$C(\theta_1,\theta_2) := \max_{\substack{j\in\{1,2,3\}\\ 0\leq\alpha\leq\theta_j}} \frac{\sin(\theta_j-\alpha)}{\sin(\theta_{j+1})} + \frac{\sin(\alpha)}{\sin(\theta_{j-1})} + \min\Big(\frac{\sin(\alpha)}{\sin(\theta_{j-1})} + \frac{\sin(\alpha+\theta_{j-1})}{\sin(\theta_{j+1})}, \frac{\sin(\theta_j-\alpha)}{\sin(\theta_{j+1})} + \frac{\sin(\alpha+\theta_{j-1})}{\sin(\theta_{j-1})} \Big).$$

*School of Computer Science, Carleton University, jit@scs.carleton.ca

†School of Electrical Engineering and Computer Science, University of Ottawa, jdecaruf@uottawa.ca

‡School of Electrical Engineering and Computer Science, University of Ottawa, jstua022@uottawa.ca

2 Preliminaries

We will denote the line segment from point u to point v as uv, and the length of uv is denoted $|uv|$. For two vertices u, v in a geometric graph G, the length of the shortest path from u to v in G is denoted $d_G(u, v)$. Then for a constant $c \geq 1$, G is said to be a c-spanner if for all vertices u, v in G, we have $d_G(u, v) \leq c|uv|$. The spanning ratio of G is the least c for which G is a c-spanner. The spanning ratio of a class of graphs $\mathcal{G}$ is the least c for which all graphs in $\mathcal{G}$ are c-spanners. A *constant* spanner is a c-spanner where c is a constant.

In a geometric graph, each vertex is identified with its coordinates. Here, one unit of memory is either a point in $\mathbb{R}^2$, or $\log_2(n)$ bits. The k-neighbourhood of a vertex u in a graph is defined to be all the vertices v such that there is a path from u to v consisting of k or fewer edges. Formally, a k-local, m-memory routing algorithm is a function that takes as input $(s, N_k(s), t, M)$, and outputs a vertex p where s is the current vertex, $N_k(s)$ is the k-neighbourhood of s, t is the destination, M is an m-unit memory register, and $p \in N_1(s)$. An algorithm is said to be c-competitive for a family of geometric graphs $\mathcal{G}$ if the path output by the algorithm for any pair of vertices $s, t \in V(G)$ for $G \in \mathcal{G}$ has length at most $c|st|$. The routing ratio of an algorithm is the least c for which the algorithm is c-competitive for $\mathcal{G}$.

Throughout this paper, we fix a triangle $\triangle$ in the plane with angles $\theta_1 \leq \theta_2 \leq \theta_3$. We assume that the corresponding corners of $\triangle$ are labelled τ_1, τ_2, τ_3. In order to keep notation cleaner, we use arithmetic modulo 3 for operations on index i when referring to corners of triangles. For example, $\tau_4 = \tau_1$, and $\tau_0 = \tau_3$. By convention, the expression $\angle abc$ will refer to the smaller angle among the clockwise and counterclockwise angles between ab and bc for three non-collinear points $a, b, c \in \mathbb{R}^2$.

For any two points u, v in the plane, define the triangle $T^{u,v}$ to be the smallest scaled translate of $\triangle$ with u and v on its boundary. Note that *smallest* implies that at least one of u, v is on a corner of $T^{u,v}$. We use $\tau_i^{u,v}$ to refer to the corner of triangle $T^{u,v}$ corresponding to τ_i. Now we define the cones, depicted in Figure 1. In particular, for a point p and index $i \in \{1, 2, 3\}$, let $C_{p,i} := \{v \in \mathbb{R}^2 | p = \tau_i^{p,v}\}$ be the positive cone centred at point p corresponding to τ_i. On the other hand, define the negative cone $\overline{C_{p,i}} := \{v \in \mathbb{R}^2 | v = \tau_i^{p,v}\}$. Note that $\overline{C_{p,i}}$ is $C_{p,i}$ rotated by π radians about p.

The TD-Delaunay graph of a vertex set $S \subseteq \mathbb{R}^2$ has an edge between vertices u and v when there exists an equilateral triangle with u, v on its boundary and no other points of S in its interior. Note that the equilateral triangle is a scaled translate of the TD unit circle. As with any Delaunay graph based on a convex distance function, every bounded face is a triangle [4]. To define the $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph, we replace the equilateral triangle with $\triangle$ containing angles of $\theta_1, \theta_2, \theta_3$. Equivalently, if F is the affine transformation that brings $\triangle$ to the equilateral triangle (the unit circle in the triangle distance), then there is an edge uv in the $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph of a set $S \subseteq \mathbb{R}^2$ exactly when $F(u)F(v)$ is an edge of the TD-Delaunay graph of $F(S)$. This alternative definition immediately leads to a local routing strategy for the $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph of a point set S: use the existing routing algorithm from [5] on the TD-Delaunay graph of $F(S)$. In Section 4.2, we show that this approach is not optimal.

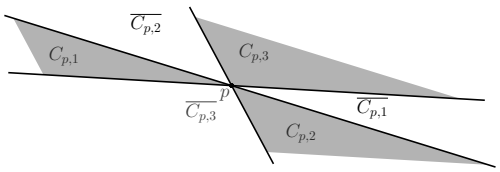


Figure 1: $C_{p,1}, C_{p,2}, C_{p,3}$ are the positive cones of p and $\overline{C_{p,1}}, \overline{C_{p,2}}, \overline{C_{p,3}}$ are the negative cones of p.

Bonichon et al. [2] showed that the TD-Delaunay graph corresponds to the half-θ_6-graph. Analogous to the half-θ_6-graph, Lubiw and Mondal [10] define the 3-sweep graph, which directly corresponds to the $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph. The 3-sweep graph G gets its name from an alternative, yet equivalent, construction. For each vertex u and each positive cone $C_{u,i}$, include in G the edge to the *nearest* vertex $v \in C_{u,i}$. By *nearest*, we mean that the triangle $T^{u,v}$ is minimal among $\{T^{u,v'} \mid v' \in C_{u,i}\}$. In this way, one can picture the leading edge $\tau_{i-1}^{u,v}\tau_{i+1}^{u,v}$ *sweeping* through cone $C_{u,i}$. Throughout the paper, we assume that no two points lie on a line parallel to a cone boundary. This ensures that each vertex has at most one neighbour in each positive cone.

One desirable property of paths is angle monotonicity. A path is angle monotone with width α if the vector of each edge on the path lies in a cone with apex angle α. Lubiw and Mondal show that the 3-sweep graph has certain angle-monotone properties which are used to upper bound the spanning ratio, see Observation 1.

Observation 1 *An angle monotone path from s to t with width α has length at most $\frac{|st|}{\cos(\alpha/2)}$ [10].*

In [10], Lubiw and Mondal also define a k-layered 3-sweep graph by combining k copies of rotated 3-sweep graphs, and provide a local routing algorithm that finds angle monotone paths in k-layered 3-sweep graphs. Note that since k is at least 4, their routing algorithm does not apply to $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs.

2.1 Our Contributions

In Section 3, we prove that $\frac{1}{\sin(\theta_1/2)}$ is a lower bound on the spanning ratio of $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs which

matches the best known upper bound [10]. Then in Section 4, we provide a lower bound on the routing ratio by showing that there exist $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs for which the routing ratio of any k-local routing algorithms is at least as large as $C(\theta_1, \theta_2)$. Then, we show that our lower bound is tight by providing an online local routing algorithm for $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs with a routing ratio of $C(\theta_1, \theta_2)$. Finally, in Section 4.2, we compare our optimal routing algorithm to the previously best-known approach to routing in $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs.

3 Spanning Ratio

We present a lower bound in the following proposition.

Proposition 1 *There exists a set of points $S \subseteq \mathbb{R}^2$ such that the TD_{θ_1,θ_2}-Delaunay graph of S has a spanning ratio of exactly $\frac{1}{\sin(\theta_1/2)} - \epsilon$ for any $\epsilon > 0$.*

Proof. We will construct a point set $S = \{a, b, \tau_1, \tau_2, \tau_3\}$ such that $d_G(a, b)$ approaches $\frac{|ab|}{\sin(\theta_1/2)}$, where G is the $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph of S. See Figure 2. Place two points a, b outside $\triangle$ each at a distance $\frac{\min(|\tau_1\tau_2|,|\tau_1\tau_3|)}{2}$ from τ_1, with a arbitrarily close to $\tau_1\tau_2$ and b arbitrarily close to $\tau_1\tau_3$. By construction of S, G has edges $\tau_1\tau_2$, $\tau_2\tau_3$, $\tau_1\tau_3$, $\tau_1 a$, $\tau_2 a$, $\tau_1 b$ and $\tau_3 b$.

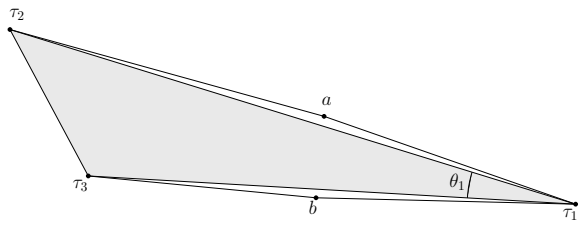


Figure 2: The shortest path from a to b passes through τ_1 in the $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graph G of the point set $\{a, b, \tau_1, \tau_2, \tau_3\}$. As a, b get closer to $\triangle$, then $d_G(a, b)$ approaches $\frac{|ab|}{\sin(\theta_1/2)}$.

The shortest path in G from a to b passes through τ_1, meaning the spanning ratio is at least $\frac{|a\tau_1|+|\tau_1 b|}{|ab|}$. This value can be made arbitrarily close to $\frac{1}{\sin(\theta_1/2)}$ as a and b move closer to the boundary of $\triangle$. While this point set may not be in general position, the vertices can be perturbed to satisfy the general position constraint. □

The upper bound of $\frac{1}{\sin(\theta_1/2)}$ follows from Lemma 6 of [10] by Lubiw and Mondal.

4 Local Routing

Local routing has been studied in many contexts, and in Section 4.2, we will show that the known routing algorithms do not give optimal results in $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs. In this section, we provide an optimal local routing algorithm. Our approach is to generalize the algorithm from [5], leading to our algorithm (refer to Algorithm 1). The key algorithmic insight lies in the threshold for making decisions in routing. Each decision is carefully made to reduce the total path length. The goal of this section is to prove the following theorem.

Theorem 2 *The routing ratio of the TD_{θ_1,θ_2}-Delaunay graph is at most $C(\theta_1, \theta_2)$. Furthermore, this bound is tight in the worst case.*

We will start with the following proposition:

Proposition 3 *Let k be a positive integer. Every k-local routing algorithm for TD_{θ_1,θ_2}-Delaunay graphs must have a routing ratio at least $C(\theta_1, \theta_2) - \epsilon$ for any $\epsilon > 0$.*

Proof. We will construct two vertex sets S_1 and S_2 and refer to their corresponding $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs as G_1 and G_2. Importantly, the k-neighbourhoods of G_1 and G_2 around the start vertices s are identical, however the rest of the graphs will be vastly different. In this way any algorithm that performs well for one graph will not for the other. These are analogous to the constructions of Figure 12 in [5]. Assume $j = 3$

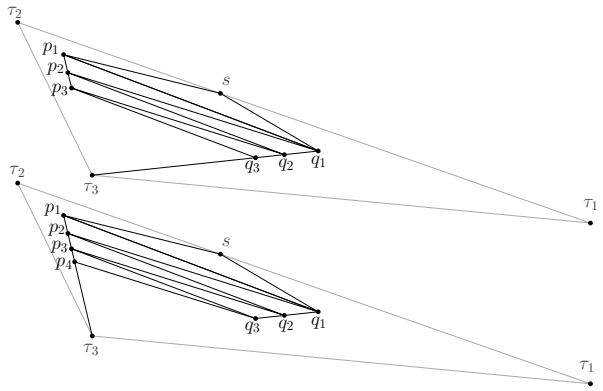


Figure 3: The $\mathrm{TD}_{\theta_1,\theta_2}$-Delaunay graphs G_1 and G_2 constructed for the lower bound of k-local routing from s to τ_3. In this example, $k = 3$.

maximizes the expression of $C(\theta_1, \theta_2)$. Let s be on $\tau_1\tau_2$. Place p_1 inside $C_{s,1} \cap C_{\tau_2,2}$ arbitrarily close to τ_2, then place q_1 in $C_{s,2} \cap C_{p_1,2} \cap C_{\tau_1,1}$ arbitrarily close to τ_1. Next, place p_2 on segment $\tau_3 p_1$ in cone $C_{q_1,1}$, arbitrarily close to p_1. Next, for $i = 2, ..., k$, place q_i such that triangle τ_3, p_i, q_i is similar to τ_3, p_1, q_1, and place p_{i+1} such that triangle τ_3, p_{i+1}, q_i is similar to τ_3, p_2, q_1. Finally, let $S_1 = \{s, p_1, ..., p_k, q_1, ..., q_k, \tau_3\}$ and $S_2 = \{s, p_1, ..., p_k, p_{k+1}, q_1, ..., q_k, \tau_3\}$. This construction ensures that G_1 contains the edges $sp_1, sq_1, p_1q_1, p_{i-1}p_i, q_{i-1}q_i, q_{i-1}p_i, p_iq_i, q_k\tau_3$

where $i \in \{2, ..., k\}$. On the other hand, G_2 contains the edges $sp_1, sq_1, p_1q_1, p_{i-1}p_i, q_{i-1}q_i, q_{i-1}p_i, p_iq_i, p_kp_{k+1}, q_kp_{k+1}, p_{k+1}\tau_3$ where $i \in \{2, ..., k\}$. Importantly, G_1 does not contain the edge τ_3p_k because q_k is the closest neighbour to τ_3 in the cone $C_{\tau_3,3}$. Similarly, G_2 does not contain the edge τ_3q_k because p_{k+1} is the closest neighbour to τ_3 in the cone $C_{\tau_3,3}$. Similarly, the edges $p_k\tau_3$ and $q_k\tau_3$ do not exist in G_1 and G_2, respectively, since τ_3 is in a negative cone of p_k and q_k.

Since the k-neighbourhood of s in G_1 and G_2 is $\{s, p_1, ..., p_k, q_1, ..., q_k\}$, then any algorithm routing from s to τ_3 will choose the same first vertex (p_1 or q_1) in G_1 and G_2. Moreover, τ_3 only has one neighbour in each graph, so any path from s to τ_3 must pass through q_k in G_1 and through p_{k+1} in G_2. Then any algorithm that visits p_1 first will output a path from s to τ_3 in G_1 of length at least $|sp_1| + |p_1q_k| + |q_k\tau_3|$. On the other hand, any algorithm that chooses to visit q_1 first will output a path from s to τ_3 in G_2 having length at least $|sq_1| + |q_1p_{k+1}| + |p_{k+1}\tau_3|$. Since p_1 is arbitrarily close to τ_2, p_2 is arbitrarily close to p_1, and q_1 is arbitrarily close to τ_1, then each p_i is arbitrarily close to τ_2 and each q_i is arbitrarily close to τ_1. Then, for any $\epsilon > 0$, the routing ratio of any algorithm is at least

$$\frac{\min(|s\tau_2| + |\tau_2\tau_1| + |\tau_1\tau_3|, |s\tau_1| + |\tau_1\tau_2| + |\tau_2\tau_3|)}{|s\tau_3|} - \epsilon$$
$$= \frac{|s\tau_1|}{|s\tau_3|} + \frac{|s\tau_2|}{|s\tau_3|} + \min\left(\frac{|s\tau_2|}{|s\tau_3|} + \frac{|\tau_1\tau_3|}{|s\tau_3|}, \frac{|s\tau_1|}{|s\tau_3|} + \frac{|\tau_2\tau_3|}{|s\tau_3|}\right) - \epsilon.$$

Finally, we obtain $C(\theta_1, \theta_2) - \epsilon$ by the law of sines in triangles $s\tau_2\tau_3$ and $\tau_1s\tau_3$, where angle $\alpha := \angle\tau_2\tau_3s$,

$$\frac{|s\tau_2|}{\sin(\alpha)} = \frac{|s\tau_3|}{\sin(\theta_2)} = \frac{|\tau_2\tau_3|}{\sin(\pi - \alpha - \theta_2)},$$
$$\frac{|s\tau_1|}{\sin(\theta_3 - \alpha)} = \frac{|\tau_1\tau_3|}{\sin(\alpha + \theta_2)} = \frac{|s\tau_3|}{\sin(\theta_1)}.$$

□

4.1 Local Routing Algorithm

In this section, we present Algorithm 1 which is a 1-local, 0-memory routing algorithm for TD_{θ_1,θ_2}-Delaunay graphs. It is generalized from the routing algorithm by Bose et al. [5]. Let s be the start vertex, t be the target vertex, and p be the current vertex. At each step of Algorithm 1, the next vertex is chosen based on the four cases (i), (ii), (iii), or (iv). To ease notation for cases (ii), (iii), and (iv), we will define the left, middle, and right regions of p: X_L, X_M, and X_R respectively, pictured in Figure 4. When t lies in a negative cone $\overline{C_{p,i}}$, then let $X_L := C_{p,i-1} \cap T^{p,t}$, $X_R := C_{p,i+1} \cap T^{p,t}$, and $X_M := \overline{C_{p,i}} \cap T^{p,t}$.

In short, the algorithm prefers to route in the region towards t, however when this is not possible, it stays

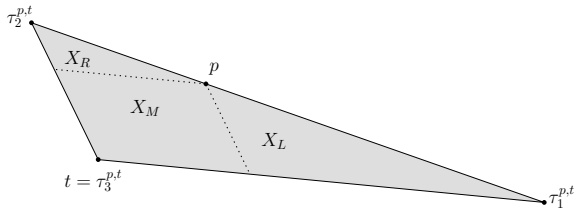


Figure 4: $T^{p,t}$ is the smallest scaled translate of $\triangle$ with p and t on its boundary.

Algorithm 1: Local Routing algorithm in TD_{θ_1,θ_2}-Delaunay graph G

Data: Two points $s, t \in S$
Result: Path in G from s to t
$p \leftarrow s$;
while $p \neq t$ **do**
Choose the next vertex v based on the following cases, then set $p \leftarrow v$

(i) Case: *t lies in a positive cone $C_{p,i}$.*
Follow the unique edge pv in $C_{p,i}$.

(ii) Case: *t lies in a negative cone $\overline{C_{p,i}}$, and both regions X_L and X_R are empty*
Let $j \in \{1, -1\}$ minimize $|p\tau_{i+j}^{p,t}| + |\tau_{i+j}^{p,t}t|$. Choose the neighbour in X_M closest to $C_{p,i+j}$ in cyclic order about p.

(iii) Case: *t lies in a negative cone $\overline{C_{p,i}}$, and only one region of $\{X_L, X_R\}$ is empty.*
If p has neighbours in X_M, choose the neighbour v closest to the empty region in cyclic order about p. Otherwise, choose the unique neighbour in the non-empty region.

(iv) Case: *t lies in a negative cone $\overline{C_{p,i}}$, and neither X_L nor X_R is empty.*
If p has neighbours in X_M, choose an arbitrary one. Otherwise, let $j \in \{1, -1\}$ minimize $|p\tau_{i+j}^{p,t}| + |\tau_{i-j}^{p,t}t|$, and choose v in $C_{p,i+j}$.

end

close to a neighbouring empty region or the side that minimizes a possible detour. Now we will prove the following upper bound:

Proposition 4 *Let s, t be two vertices in a TD_{θ_1,θ_2}-Delaunay graph G. When t is in a negative cone of s, then the path $P_{s,t}$ output by Algorithm 1 from s to t in G has ratio $\frac{|P_{s,t}|}{|st|}$ at most $C(\theta_1, \theta_2)$. When t is in a positive cone of s, then $\frac{|P_{s,t}|}{|st|}$ is at most $\frac{1}{\sin(\theta_1/2)}$.*

Notice that when the angles $\theta_1, \theta_2, \theta_3$ are all equal to $\frac{\pi}{3}$, then $C(\theta_1, \theta_2)$ in Proposition 3 for routing in a negative cone reaches a maximum of $5/\sqrt{3}$ when $\alpha = \frac{\pi}{6}$, matching the bound from [5]. Furthermore, the expression for routing in a positive cone matches the spanning ratio.

Proof. We will bound the path chosen by Algorithm 1 by defining a potential for each vertex along a path and showing that at each step, the potential drops by at least the length of the chosen edge. We define the potential as follows, depicted in Figure 5.

- Case (i): $\Phi(p,t) := \max_{j=\pm 1}(|p\tau_{i+j}^{p,t}| + |\tau_{i+j}^{p,t}t|)$.
- Case (ii): $\Phi(p,t) := \min_{j=\pm 1}(|p\tau_{i+j}^{p,t}| + |\tau_{i+j}^{p,t}t|)$.
- Case (iii): $\Phi(p,t) := |p\tau_{i+j}^{p,t}| + |\tau_{i+j}^{p,t}t|$ where the empty region (X_L or X_R) is $C_{p,i+j} \cap T^{p,t}$.
- Case (iv): $\Phi(p,t) := \min_{j=\pm 1}(|p\tau_{i+j}^{p,t}| + |\tau_{i+j}^{p,t}\tau_{i-j}^{p,t}| + |\tau_{i-j}^{p,t}t|)$.

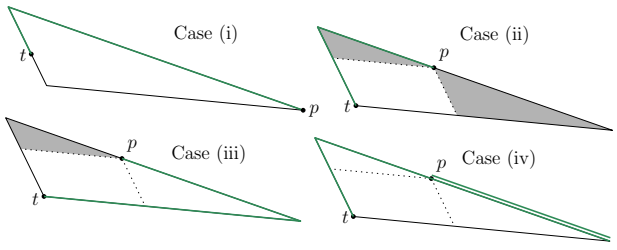


Figure 5: The potential is given by the green path. In this example, i takes values $1, 3, 3, 3$ for cases (i), (ii), (iii), (iv) respectively. The grey regions are empty.

Now, we will show that in each case of Algorithm 1, the length of each chosen edge is less than the drop in potential. More precisely, we want to show $|pv| + \Phi(v,t) \leq \Phi(p,t)$ for cases (i),(ii),(iii), and (iv).

Suppose the current vertex is p and the case is (i), as can be seen in Figure 6. Then after an edge pv is chosen, the current vertex will proceed to v and the case will be either (i), (ii), or (iii). Case (iv) is not possible when t lies in a negative cone of v because at least one of the regions of v is empty. Then the next potential, $\Phi(v,t)$, passes through some vertex $\tau_{i+k}^{v,t}$ for $k = \pm 1$. We have

$$\begin{aligned}|pv| + \Phi(v,t) &\leq (|p\tau_{i+k}^{p,v}| + |\tau_{i+k}^{p,v}v|) + (|v\tau_{i+k}^{v,t}| + |\tau_{i+k}^{v,t}t|)\\ &= (|p\tau_{i+k}^{p,v}| + |v\tau_{i+k}^{v,t}|) + (|\tau_{i+k}^{p,v}v| + |\tau_{i+k}^{v,t}t|)\\ &= |p\tau_{i+k}^{p,t}| + |\tau_{i+k}^{p,t}t|\\ &\leq \max_{j=\pm 1}(|p\tau_{i+j}^{p,t}| + |\tau_{i+j}^{p,t}t|) = \Phi(p,t)\end{aligned}$$

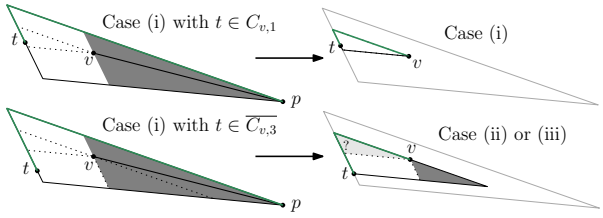


Figure 6: Bounding the potential in case (i) since t lies in $C_{p,1}$. The dark grey regions are empty.

Next, suppose the current vertex is p and the case is (ii), depicted in Figure 7. Let j minimize the expression from $\Phi(p,t)$. Notice that since we choose the edge closest in cyclic order about p to the region $C_{p,i+j} \cap T^{p,t}$, then we can deduce that v has no neighbours in its region $C_{v,i+j} \cap T^{v,t}$. Therefore once the current vertex proceeds to v, then the possible cases are only (ii) or (iii). Then we have

$$\begin{aligned}|pv| + \Phi(v,t) &\leq (|p\tau_{i+j}^{p,v}| + |\tau_{i+j}^{p,v}v|) + (|v\tau_{i+j}^{v,t}| + |\tau_{i+j}^{v,t}t|)\\ &= (|p\tau_{i+j}^{p,v}| + |v\tau_{i+j}^{v,t}|) + (|\tau_{i+j}^{p,v}v| + |\tau_{i+j}^{v,t}t|)\\ &= |p\tau_{i+j}^{p,t}| + |\tau_{i+j}^{p,t}t| = \Phi(p,t)\end{aligned}$$

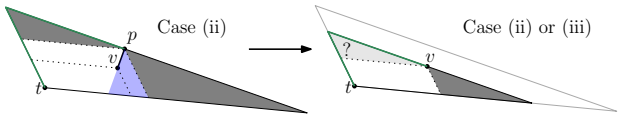


Figure 7: Case (ii) when t lies in $\overline{C_{p,3}}$ and $j = -1$. The dark grey and blue regions are empty.

Now suppose p is the current vertex, the case is (iii), and $C_{p,i+j} \cap T^{p,t}$ is the empty region, shown in Figure 8. If X_M is not empty, then the choice of closest neighbour v to $C_{p,i+j} \cap T^{p,t}$ guarantees that the corresponding region $C_{v,i+j} \cap T^{v,t}$ of v is also empty. Likewise, if X_M is empty, then choosing the unique neighbour in $C_{p,i-j} \cap T^{p,t}$ again guarantees that v has an empty region $C_{v,i+j} \cap T^{v,t}$. Either way, once the current vertex continues to v, then the case must be either (ii) or (iii). Then the exact same sequence of inequalities as from case (ii) completes the argument.

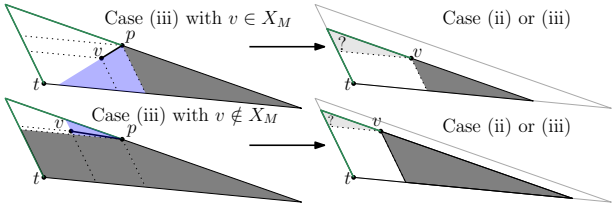


Figure 8: Case (iii) when t lies in $\overline{C_{p,3}}$ and $j = -1$. The dark grey and blue regions are empty.

Finally, we move on to case (iv), where p is the current vertex. After choosing the next vertex v, the next possible cases are (ii), (iii), or (iv). Let $j = \pm 1$ minimize the expression of $\Phi(p,t)$. In any case, we have $\Phi(v,t) \leq (|v\tau_{i+j}^{v,t}| + |\tau_{i+j}^{v,t}\tau_{i-j}^{v,t}| + |\tau_{i-j}^{v,t}t|)$ by the triangle inequality. When $v \in X_M$, then we use the following inequalities to prove the claim, also shown in Figure 9.

1. $|pv| \leq |p\tau_{i-j}^{v,p}| + |\tau_{i-j}^{v,p}v|$ by triangle inequality
2. $|\tau_{i-j}^{v,t}t| + |\tau_{i-j}^{v,p}v| = |\tau_{i-j}^{p,t}t|$ by projection
3. $|p\tau_{i-j}^{v,p}| \leq |\tau_{i+j}^{v,p}\tau_{i-j}^{v,p}|$ since p lies on $\tau_{i+j}^{v,p}\tau_{i-j}^{v,p}$
4. $|\tau_{i+j}^{v,t}\tau_{i-j}^{v,t}| + |\tau_{i+j}^{v,p}\tau_{i-j}^{v,p}| = |\tau_{i+j}^{p,t}\tau_{i-j}^{p,t}|$ by translation and projection
5. $|v\tau_{i+j}^{v,t}| \leq |p\tau_{i+j}^{t,p}|$ by projection

When v is not in X_M, then let u be the intersection of $p\tau_i^{p,v}$ and $\tau_{i-j}^{t,v}\tau_{i+j}^{t,v}$. The following inequalities suffice to prove the claim.

1. $|pv| \leq |pu| + |uv|$ by triangle inequality
2. $|uv| + |v\tau_{i+j}^{v,t}| \leq |p\tau_{i+j}^{p,t}|$ by projection
3. $|\tau_{i-j}^{v,t}t| + |pu| = |\tau_{i-j}^{p,t}t|$ by projection
4. $|\tau_{i+j}^{v,t}\tau_{i-j}^{v,t}| \leq |\tau_{i+j}^{p,t}\tau_{i-j}^{p,t}|$ by projection

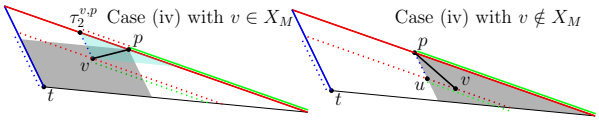


Figure 9: Bounding the potential in case (iv) when t lies in $\overline{C_{p,3}}$, with $i = 3, j = 1$. The dotted paths representing $|pv| + \Phi(v,t)$ are shorter than the corresponding solid paths of $\Phi(p,t)$. The grey region contains v, and the blue triangle is $T^{v,p}$.

Since $\Phi(t,t) = 0$, then the path from s to t output by Algorithm 1 can have length at most $\Phi(s,t)$. When t is in a positive cone of s, then the potential is defined using case (i). The corresponding path $p\tau_{i+j}^{p,t} + \tau_{i+j}^{p,t}t$ is $\pi - \theta_{i+j}$ monotone, then the routing ratio in such a case can be at most $\frac{1}{\sin(\theta_1/2)}$ by Observation 1.

On the other hand, when t is in a negative cone of s, there are three possible cases: (ii), (iii) or (iv). The triangle inequality tells us that $\Phi(s,t)$ is largest in case (iv). Then, similar to the proof of Proposition 3, the routing ratio is bounded by $C(\theta_1, \theta_2)$ using the law of sines. □

Finally, Theorem 2 is a consequence of Propositions 3 and 4 since Algorithm 1 is 1-local.

4.2 Comparison to known routing algorithms

In this subsection, we show that currently known local routing algorithms when applied on the TD_{θ_1,θ_2}-Delaunay graph are suboptimal. Firstly, note that by using a stretch factor upper bound from Section 3, we can apply the technique of Bose and Morin [6] to obtain a local routing algorithm that finds a path between any two vertices with length at most 9 times the stretch factor, which is not optimal. Another approach is to route in TD_{θ_1,θ_2}-Delaunay graphs by combining the algorithm of Bose et al. [5] with an affine transformation. When $\triangle$ is the equilateral triangle, then Algorithm 1 simplifies to the standard TD-Delaunay routing algorithm from [5]. In this case, notice that the thresholds in cases (ii) and (iv) simplify so that j refers to the corner $\tau_{i+j}^{p,t}$ nearest p. In other words, the decision threshold is the midpoint of the segment $\tau_{i+1}^{p,t}\tau_{i-1}^{p,t}$. We will analyze this standard TD-Delaunay routing algorithm when it is used on the affine transformation of a general TD_{θ_1,θ_2}-Delaunay graph. Since affine transformations preserve midpoints, then the decision threshold in case (iv) is also the midpoint of the segment $\tau_{i+1}^{p,t}\tau_{i-1}^{p,t}$. It is in this way that applying an affine transformation to the existing algorithm differs from our Algorithm 1. To see the difference in routing ratio of these two approaches, consider the construction of G_1 from Proposition 3. If we enforce $|s\tau_2| < |s\tau_1|$, then the path output by the affine transformation of the standard TD-Delaunay routing algorithm would choose to visit p_1 first. The routing ratio of this algorithm would therefore be at least

$$\frac{\sin(\theta_3 - \alpha)}{\sin(\theta_1)} + \frac{\sin(\alpha)}{\sin(\theta_2)} + \frac{\sin(\alpha)}{\sin(\theta_2)} + \frac{\sin(\alpha + \theta_2)}{\sin(\theta_1)} - \epsilon$$

where $\alpha := \angle\tau_2\tau_3 s$. For example, when $\theta_1 = \frac{\pi}{6}$, $\theta_2 = \frac{\pi}{5}$, and $\alpha = \frac{\pi}{3}$ then the routing ratio of the standard TD-Delaunay algorithm under an affine transformation is strictly more than 6.55, whereas the optimal routing ratio is less than 6.52 by Proposition 4.

References

[1] N. Bonichon, P. Bose, J.-L. De Carufel, L. Perkovic, and A. van Renssen. Upper and lower bounds for online

routing on Delaunay triangulations. *Discret. Comput. Geom.*, 58(2):482–504, 2017.

[2] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *WG*, volume 6410 of *Lecture Notes in Computer Science*, pages 266–278, 2010.

[3] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perkovic. Tight stretch factors for l_1- and l_∞-Delaunay triangulations. *Comput. Geom.*, 48(3):237–250, 2015.

[4] P. Bose, P. Carmi, S. Collette, and M. H. M. Smid. On the stretch factor of convex Delaunay graphs. *J. Comput. Geom.*, 1(1):41–56, 2010.

[5] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Optimal local routing on Delaunay triangulations defined by empty equilateral triangles. *SIAM J. Comput.*, 44(6):1626–1649, 2015.

[6] P. Bose and P. Morin. Competitive online routing in geometric graphs. *Theor. Comput. Sci.*, 324(2-3):273–288, 2004.

[7] P. Chew. There is a planar graph almost as good as the complete graph. In *SCG*, pages 169–177. ACM, 1986.

[8] P. Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39(2):205–219, 1989.

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

[10] A. Lubiw and D. Mondal. Construction and local routing for angle-monotone graphs. *J. Graph Algorithms Appl.*, 23(2):345–369, 2019.

[11] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[12] S. Njoo. The exact spanning ratio of the parallelogram Delaunay graph, *Mater's Thesis, University of Ottawa*, 2024.

[13] L. Perkovic, M. Dennis, and D. Türkoglu. The stretch factor of hexagon-Delaunay triangulations. *J. Comput. Geom.*, 12(2):86–125, 2021.

[14] A. van Renssen, Y. Sha, Y. Sun, and S. Wong. The tight spanning ratio of the rectangle Delaunay triangulation. In *ESA*, volume 274 of *LIPIcs*, pages 99:1–99:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[15] G. Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM J. Comput.*, 42(4):1620–1659, 2013.

[16] G. Xia and L. Zhang. Toward the tight bound of the stretch factor of Delaunay triangulations. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, Toronto, Ontario, Canada, August 10-12*, 2011.

Exact solutions to the Weighted Region Problem*

Sarita de Berg† Guillermo Esteban‡ Rodrigo I. Silveira§ Frank Staals¶

Abstract

In this paper, we consider the Weighted Region Problem. In the Weighted Region Problem, the length of a path is defined as the sum of the weights of the subpaths within each region, where the weight of a subpath is its Euclidean length multiplied by a weight $\alpha \geq 0$ depending on the region. We study a restricted version of the problem of determining shortest paths through a single weighted rectangular region. We prove that even this very restricted version of the problem is unsolvable within the Algebraic Computation Model over the Rational Numbers (ACM$\mathbb{Q}$). On the positive side, we provide the equations for the shortest paths that are computable within the ACM$\mathbb{Q}$. Additionally, we provide equations for the bisectors between regions of the Shortest Path Map for a source point on the boundary of (or inside) the rectangular region.

1 Introduction

The Weighted Region Problem (WRP) [15] is a well-known geometric problem that, despite having been studied extensively, is still far from being well understood. Consider a subdivision of the plane into (usually polygonal) regions. Each region R_i has a weight $\alpha_i \geq 0$, representing the cost per unit distance of traveling in that region. Thus, a straight-line segment σ, of Euclidean length $|\sigma|$, between two points in the same region has weighted length $\alpha_i \cdot |\sigma|$ when traversing the interior of R_i, or $\min\{\alpha_i, \alpha_j\} \cdot |\sigma|$ if it goes along the edge between R_i and R_j. Then, the weighted length of a path through a subdivision is the sum of the weighted lengths of its subpaths through each face or edge. The resulting metric is called the *Weighted Region Metric*. The WRP entails computing a shortest path $\pi(s,t)$ between two given points s and t under this metric. We denote the weighted length of $\pi(s,t)$ by $d(s,t)$. Figure 1 shows how the shape of a shortest path changes as the weight of one region varies.

Existing algorithms for the WRP—in its general formulation—are approximate. Since the seminal work by Mitchell and Papadimitriou [15], with the first $(1+\varepsilon)$-approximation, several algorithms have been proposed, with improvements on running times, but always keeping some dependency on the vertex coordinates sizes and weight ranges. These methods are usually based on the continuous Dijkstra's algorithm, subdividing triangle edges in parts for which crossing shortest paths have the same combinatorial structure (e.g., [15]), or on adding Steiner points (e.g., see [1, 2, 3, 5, 18]). However, rather recently it has been proved that computing an exact shortest path between two points using the Weighted Region Metric, even if there are only three different weights, is an unsolvable problem in the Algebraic Computation Model over the Rational Numbers (ACM$\mathbb{Q}$) [6]. In the ACM$\mathbb{Q}$ one can compute exactly any number that can be obtained from rational numbers by applying a finite number of operations from $+, -, \times, \div$, and $\sqrt[k]{}$, for any integer $k \geq 2$. This provides a theoretical explanation for the lack of exact algorithms for the WRP, and justifies the study of approximation methods.

This also raises the question of which are the special cases for which the WRP can be solved exactly. Two natural ways to restrict the problem are by limiting the possible weights and by restricting the shape of the regions. For example, computing a shortest path among polygonal or curved obstacles can be seen as a variant of the WRP with weights in the set $\{1, \infty\}$. Efficient algorithms exists for this problem, culminating with the recent algorithms by Wang [19] for polygonal obstacles, and by Hershberger et al. [11] for shortest paths among curved obstacles. The case for polygonal regions with weights in $\{0, 1, \infty\}$ can be solved in $O(n^2)$ time [9] by constructing a graph known as the *critical graph*, an extension of the visibility graph. Other variants that can be solved exactly correspond to regions shaped as regular k-gons with weight ≥ 2 (since they can be considered as obstacles), or regions with two weights $\{1, \alpha\}$ consisting of parallel strips [16]. In the latter case, the angle of incidence in each of the strips is the same, so they can be rearranged so that they are all together, and the angle of incidence can be computed exactly using Snell's law of refraction.

*Work by G. E. and R. I. S. has been supported by project PID2019-104129GB-I00 funded by MICIU/AEI/10.13039/501100011033. G. E. was also funded by an FPU of the Universidad de Alcalá.

†Department of Information and Computing Sciences, Utrecht University, s.deberg@uu.nl

‡Departamento de Física y Matemáticas, Universidad de Alcalá and School of Computer Science, Carleton University, g.esteban@uah.es

§Departament de Matemàtiques, Universitat Politècnica de Catalunya, rodrigo.silveira@upc.edu

¶Department of Information and Computing Sciences, Utrecht University, f.staals@uu.nl

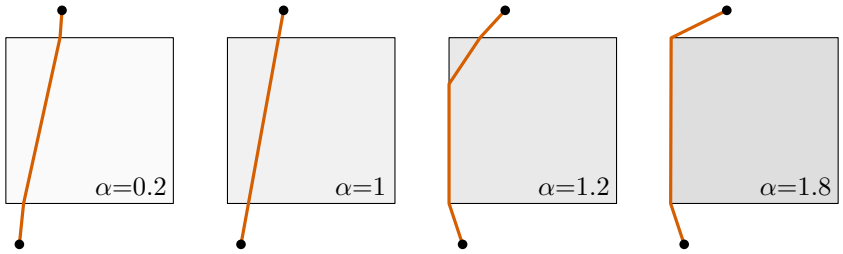


Figure 1: Examples of shortest paths between two points—shown in orange—for two weighted regions. The unbounded region has weight 1, the squares have varying weight α.

Our results. In light of the fact that the WRP is unsolvable in the ACM$\mathbb{Q}$ already for three different weights, in this work we study the case of two arbitrary weights, that is, weights in $\{1, \alpha\}$, where $\alpha \in \mathbb{Q}^+$. In particular, and without loss of generality, we assume that the weight of the unbounded region is 1. Otherwise, we could always rescale the weights to be 1 outside the regions. This case is particularly interesting, since an algorithm for weights $\{1, \alpha\}$ can be transformed into one for weights in $\{0, 1, \alpha, \infty\}$ [13]. However, the variant with weights $\{1, \alpha\}$ was conjectured to be as hard as the general WRP problem, see the first open problem in [9, Section 7]. (The results in [6] do not directly apply to weights $\{0, 1, \alpha, \infty\}$.)

This paper is organized as follows. First we present some preliminaries in Section 2. In Section 3 we consider two weights and one rectangular region R, with the source point s on its boundary or inside. For this setting, we figure out all types of possible optimal paths and give exact formulas to compute their lengths. In Section 3.3 we focus on the case where s is outside of R, and prove that in this case the WRP with weights $\{1, \alpha\}$ is already unsolvable in the ACM$\mathbb{Q}$, confirming the suspicions of Mitchell [13]. In Section 4 we explore the computation of the Shortest Path Map for s. We finish with some conclusions in Section 5.

2 Shortest paths and their properties

In this section we briefly review some key properties of shortest paths in weighted regions.

First, with our assumption that the weight within each region does not account for the effect of certain force fields that favors some directions of travel, shortest paths in the Weighted Region Problem will always be piecewise linear, see [15, Lemma 3.1]. Second, it is known that shortest paths must obey Snell's law of refraction. So we can think of a shortest path as a ray of light. Throughout this paper, the *angle of incidence* θ is defined as the minimum angle between the incoming ray and the vector perpendicular to the region boundary. The *angle of refraction* θ' is defined as the minimum angle between the outgoing ray and the vector perpendicular to the region boundary. Snell's law states that whenever the ray goes from one region R_i to another region R_j, then $\alpha_i \sin\theta = \alpha_j \sin\theta'$. In addition, whenever $\alpha_i > \alpha_j$, the angle θ_c at which $\frac{\alpha_i}{\alpha_j} \sin\theta_c = 1$ is called the *critical angle*. A ray that hits an edge at an angle of incidence greater than θ_c, will be totally reflected from the point at which it hits the boundary. In our problem, a shortest path will never be incident to an edge at an angle greater than θ_c.

Finally, if the space only contains orthoconvex regions[1] with weight at least $\sqrt{2}$, they can be simply considered as obstacles [16]. Thus, since we focus on a rectangular region R, we assume that its weight is $0 < \alpha < \sqrt{2}$. However, first we provide some general properties of shortest paths for arbitrary weighted regions that are interesting on their own.

Lemma 1 *Let $\mathcal{S}$ be a polygonal subdivision for which each region has a weight in the set $\{1, \alpha\}$, with $\alpha \geq 0$. A shortest path $\pi(s, t)$ visits any edge of the subdivision at most once.*

Proof. Assume, for the sake of contradiction, that $\pi(s, t)$ intersects an edge e in at least two disjoint intervals I_1 and I_3 (note that I_1 and I_3 could be points). Moreover, let $p_1 \in I_1$ and $p_3 \in I_3$ be points for which the subpath $\pi(p_1, p_3) \subseteq \pi(s, t)$ does not intersect e in any points other than p_1 and p_3. Let p_2 be a point on $\pi(p_1, p_3)$ between p_1 and p_3, which thus does not lie on e. Now observe that there exists a path $\overline{p_1 p_3}$ from p_1 to p_3 of length $\min\{1, \alpha\}|\overline{p_1 p_3}|$. Since p_2 does not lie on $\overline{p_1 p_3}$, it follows by the triangle inequality that the length of $\pi(p_1, p_3)$ is strictly larger than $\min\{1, \alpha\}|\overline{p_1 p_3}|$. Hence, $\pi(s, t)$ is not a shortest path, and we obtain a contradiction. □

Observe that the previous result is not true when there are more than two weights, see [15, Figure 2].

Corollary 2 *Let $\mathcal{S}$ be a polygonal subdivision with n vertices for which each region has a weight in the set $\{1, \alpha\}$, with $\alpha \geq 0$. Any shortest path $\pi(s, t)$ is a polygonal chain with at most $O(n)$ vertices.*

[1] A region is orthoconvex if its intersection with every horizontal and vertical line is connected or empty [17].

Proof. Any shortest path is a polygonal chain whose interior vertices all lie on edges of $\mathcal{S}$, see [15, Proposition 3.8]. By Lemma 1, each edge contributes with at most two vertices. □

We observe that if the regions use only one of two weights $\{1, \alpha\}$, Corollary 2 implies that the time complexity of the algorithm proposed by Mitchell and Papadimitriou [15] can be improved by a quartic factor to $O(n^4 L)$, where L is the precision of the instance.

3 Computing a shortest path

We now consider the problem of computing a shortest path $\pi(s,t)$ from s to t when the region R is an axis-aligned rectangle of weight α. The exact shape of $\pi(s,t)$ depends on the position of s and t with respect to R, and on the value of α. In Sections 3.1 and 3.2 we consider the case that s lies on the boundary or inside of R, respectively. We categorize the various types of shortest paths, and show that we can compute the shortest path of each type, and thus we can compute $\pi(s,t)$. In Section 3.3, we consider the case that s and t lie outside R. In this case $\pi(s,t)$ may have only two vertices on the boundary of R, and these vertices may not have the critical angle property. We show that the coordinates of these vertices cannot be computed exactly within the ACM$\mathbb{Q}$.

3.1 The source point s lies on the boundary of R

Throughout this section we consider the case where s is restricted to the boundary of R, a rectangle of unit height with top-left corner at $(0,0)$. Let $s = (s_x, 0)$, $s_x > 0$, be a point on the top side of R, see Figure 2. In addition, we assume that t is to the left of the line through s perpendicular to the top side of R. The other cases are symmetric.

Shortest path types. Lemma 1 implies that in this setting, there are only $O(1)$ combinatorial types of paths that we have to consider. More precisely, we have that:

Observation 1 *Let s be a point on the top boundary of a rectangle R with weight $0 < \alpha < \sqrt{2}$. There are 12 types of shortest paths $\pi_i(s,t)$, shown in Figure 2, up to symmetries.*

Note that only some of the types can exist for both $\alpha < 1$ and $1 \le \alpha < \sqrt{2}$. These types are included twice in Figure 2, once for each regime of α.

Length of $\pi_i(s,t)$. When s is on the boundary of R, there is at most one vertex of $\pi_i(s,t)$ without the critical angle property. This allows us to compute the exact coordinates of the vertices of $\pi_i(s,t)$ in the ACM$\mathbb{Q}$. We now provide the equations for the length $d_i(s,t)$ of the 12 types of shortest paths $\pi_i(s,t)$. Theorems 3 and 4 summarize the results. The proofs of the equations, which are based on Snell's law of refraction, are deferred to Appendix A.

Theorem 3 *Let $s = (s_x, 0)$ be a point on the boundary of R with weight $0 < \alpha < \sqrt{2}$, and let $\beta = \alpha^2 - 1$. The shortest path $\pi(s,t) = \pi_i(s,t)$ from s to a point $t = (t_x, t_y)$ outside R, and its length can be computed in $O(1)$ time in the ACM$\mathbb{Q}$. In particular, the length $d(s,t) = d_i(s,t)$ is given by*

- $d_1(s,t) = \sqrt{(s_x - t_x)^2 + t_y^2}$,
- $d_2(s,t) = \alpha(s_x - t_x) + \sqrt{1-\alpha^2}\, t_y$,
- $d_3(s,t) = \alpha s_x + \sqrt{t_x^2 + t_y^2}$,
- $d_4(s,t) = s_x + \sqrt{t_x^2 + t_y^2}$,
- $d_5(s,t) = s_x - \sqrt{1-\beta}\, t_x - \sqrt{\beta}\, t_y$,
- $d_6(s,t) = \alpha\sqrt{s_x^2 + y^2} + \sqrt{t_x^2 + (t_y - y)^2}$, *where y is the unique real solution in the interval $(t_y, 0)$ to the equation*

$$\beta y^4 - 2t_y\beta y^3 + \big[\alpha^2 t_x^2 + \beta t_y^2 - s_x^2\big] y^2 + 2s_x^2 t_y y - s_x^2 t_y^2 = 0,$$

- $d_7(s,t) = \sqrt{\beta} s_x + 1 + \sqrt{t_x^2 + (t_y + 1)^2}$,
- $d_8(s,t) = \sqrt{\beta}(s_x + t_x) - \sqrt{1-\beta}(1 + t_y) + 1$,
- $d_9(s,t) = \alpha\sqrt{(s_x - x)^2 + 1} + \sqrt{(t_x - x)^2 + (t_y + 1)^2}$, *where x is the unique real solution in the interval (t_x, s_x) to the equation*

$$\begin{aligned}\beta x^4 - 2\beta(t_x + s_x)x^3 + \big[&\beta(s_x^2 + t_x^2 + 4s_x t_x) \\ &+ \alpha^2(1 + t_y)^2 - 1\big]x^2 - 2\big[\beta(t_x s_x^2 + t_x^2 s_x) \\ &+ \alpha^2(1 + t_y)^2 s_x - t_x\big]x + \beta t_x^2 s_x^2 \\ &+ \alpha^2(1 + t_y)^2 s_x^2 - t_x^2 = 0. \qquad (1)\end{aligned}$$

Theorem 4 *Let $s = (s_x, 0)$ be a point on the boundary of R with weight $0 < \alpha < \sqrt{2}$. The shortest path $\pi(s,t) = \pi_i(s,t)$ from s to a point $t = (t_x, t_y)$ inside R, and its length can be computed in $O(1)$ time in the ACM$\mathbb{Q}$. In particular, the length $d(s,t) = d_i(s,t)$ is given by*

- $d_{10}(s,t) = s_x - t_x - \sqrt{\alpha^2 - 1}\, t_y$,
- $d_{11}(s,t) = \alpha\sqrt{(s_x - t_x)^2 + t_y^2}$,
- $d_{12}(s,t) = \sqrt{\alpha^2 - 1}(s_x + t_x) - t_y$.

3.2 The source point s lies inside R

We now consider the case where s is restricted to the interior of the rectangle R.

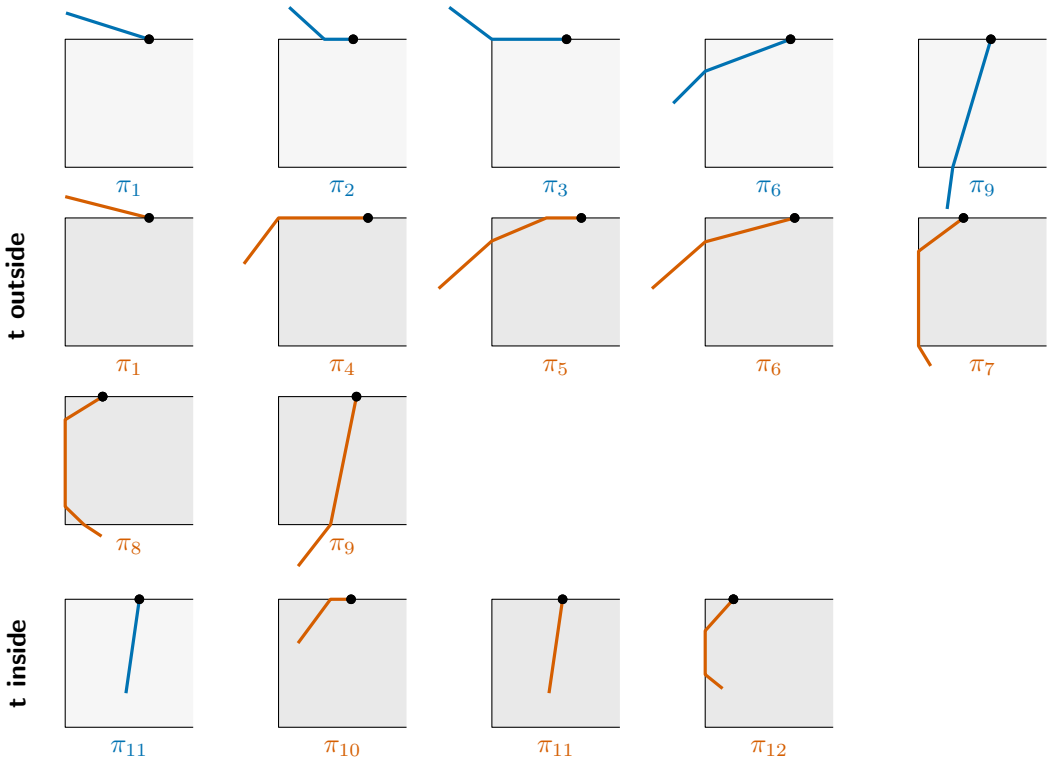


Figure 2: Path types for s on the boundary of R of weight $\alpha < 1$ (blue) and $1 \leq \alpha < \sqrt{2}$ (orange).

Observation 2 *Let s be a point in a rectangle R with weight $0 < \alpha < \sqrt{2}$. There are 6 types of shortest paths $\pi_i(s,t)$, for $i \in \{6,7,8,9,11,12\}$, up to symmetries.*

The types of shortest paths are similar to the ones defined in Observation 1, see the paths in Figure 2 where the top side of R or the region above R is not intersected. As in Theorems 3 and 4, we can thus compute (the length of) a shortest path (of each type) exactly, albeit that the expressions for the length are dependent on the location of s in R. Note that Theorems 3 and 4 give exact lengths for all path types when R has height > 1 and s is at distance exactly 1 from the bottom boundary of R.

3.3 The source point s lies outside of R

When both the source and the target point are outside of R, the shortest path can again be of many different types. In particular, the types in Figure 2 can be generalized to this setting. There are two special cases where the shortest path bends *twice*, and these two vertices do not have the critical angle property: it can bend on two opposite sides of the rectangle, or on two incident sides. In the first case, the angles at both vertices are equal, and the shortest path can be computed exactly [16]. For the second case, we show that it is not possible to compute the coordinates of the vertices exactly in the ACM$\mathbb{Q}$. Hence, the WRP limited to two weights $\{1, \alpha\}$ is not solvable within the ACM$\mathbb{Q}$. Note that this path type can occur in an even simpler setting, where R is a single quadrant instead of a rectangle.

Theorem 5 *The Weighted Region Problem with weights in the set $\{1, \alpha\}$, with $0 < \alpha < \sqrt{2}$, and $\alpha \neq 1$, cannot be solved exactly within the ACM$\mathbb{Q}$, even if R is a single quadrant.*

Proof. Consider the situation where a horizontal and a vertical line intersect at the point $O = (50, 150)$. Let R be the quadrant such that O is its top-left corner, and has weight $\alpha = 1.2$. Recall that the weight outside R is 1. Let $s = (0, 0)$ be the source point and $t = (200, 200)$ be the target point, see Figure 3. We follow the approach of De Carufel et al. [6] to show that the polynomial that represents a solution to the Weighted Region Problem in this situation is not solvable within the ACM$\mathbb{Q}$. The following lemma, which is a consequence of Theorem 1 and Lemma 2 of De Carufel et al. [6], see also [4, 7], states when a polynomial is unsolvable within the ACM$\mathbb{Q}$.

Lemma 6 *Let $p(x)$ be a polynomial of odd degree $d \geq 5$. Suppose there are three primes q_1, q_2, q_3 that do not divide the discriminant of $p(x)$, such that*

$$\begin{aligned} p(x) &\equiv p_d(x) \mod q_1, \\ p(x) &\equiv p_1(x)p_{d-1}(x) \mod q_2, \text{ and} \\ p(x) &\equiv p_2(x)p_{d-2}(x) \mod q_3, \end{aligned}$$

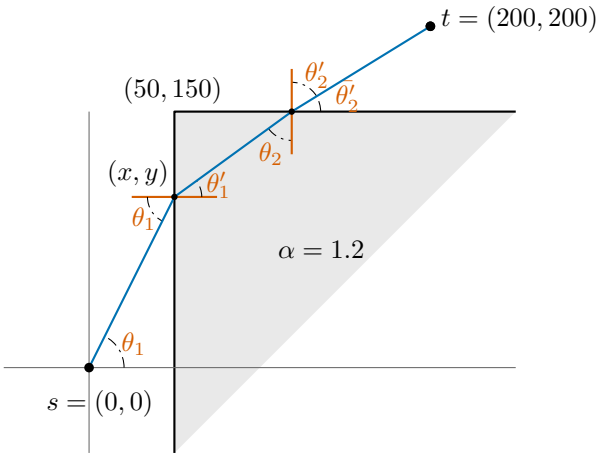


Figure 3: The set-up for the proof that even for two weights the Weighted Region Problem cannot be solved within the ACM$\mathbb{Q}$.

where $p_i(x)$ denotes an irreducible polynomial of degree i modulo the given prime. Then $p(x) = 0$ is unsolvable within the ACM$\mathbb{Q}$.

Let (x, y) be the coordinates of the intersection point of the path $\pi(s, t)$ with the vertical side of the quadrant. We denote by θ_1 the angle made by the ray from s to (x, y) with the perpendicular to the vertical side of the quadrant, by θ_1' the angle of the refracted ray with respect to the same line, and by $\bar{\theta}_2'$ the angle of the refracted ray with respect to the top side of the quadrant, see Figure 3. First we use Snell's law to show the following relations between the angles:

$$\sin\theta_1' = \frac{\sin\theta_1}{\alpha}, \tag{2}$$

$$\cos\bar{\theta}_2' = \sqrt{\alpha^2 - \sin^2\theta_1}. \tag{3}$$

To obtain the relation in Equation (3) we use Equation (2), and the fact that $\cos\bar{\theta}_2' = \alpha\cos\theta_1'$, and $\cos\theta_1' = \sqrt{1 - \sin^2\theta_1'}$. We then express the sum of the horizontal distances in terms of tangents of the angles, as follows:

$$200 = 50 + \frac{150 - y}{\tan\theta_1'} + \frac{50}{\tan\bar{\theta}_2'}$$
$$\Longrightarrow 0 = \frac{150 - y}{\tan\theta_1'} + \frac{50}{\tan\bar{\theta}_2'} - 150.$$

Using that $y = 50\tan\theta_1$, we obtain an equation only containing θ_1, θ_1' and $\bar{\theta}_2'$.

$$0 = \frac{150 - 50\tan\theta_1}{\tan\theta_1'} + \frac{50}{\tan\bar{\theta}_2'} - 150.$$

We then apply the trigonometric identities $\tan\theta = \frac{\sin\theta}{\sqrt{1-\sin^2\theta}}$, for θ_1 and θ_1', and $\tan\theta = \frac{\sqrt{1-\cos^2\theta}}{\cos\theta}$, for $\bar{\theta}_2'$.

$$0 = \frac{150 - 50\frac{\sin\theta_1}{\sqrt{1-\sin^2\theta_1}}}{\frac{\sin\theta_1'}{\sqrt{1-\sin^2\theta_1'}}} + \frac{50}{\frac{\sqrt{1-\cos^2\bar{\theta}_2'}}{\cos\bar{\theta}_2'}} - 150.$$

Finally, we replace all instances of $\sin\theta_1'$ and $\cos\bar{\theta}_2'$ by expressions in $\sin\theta_1$ using Equations (2) and (3).

$$\begin{aligned}
0 &= \frac{150 - 50\frac{\sin\theta_1}{\sqrt{1-\sin^2\theta_1}}}{\frac{\sin\theta_1}{\alpha\sqrt{1-\frac{\sin^2\theta_1}{\alpha^2}}}} + \frac{50}{\frac{\sqrt{1-(\alpha^2-\sin^2\theta_1)}}{\sqrt{\alpha^2-\sin^2\theta_1}}} - 150 \\
&= \left(150 - 50\frac{\sin\theta_1}{\sqrt{1-\sin^2\theta_1}}\right)\cdot\frac{\sqrt{\alpha^2-\sin^2\theta_1}}{\sin\theta_1} \\
&\quad + \frac{50\sqrt{\alpha^2-\sin^2\theta_1}}{\sqrt{1-(\alpha^2-\sin^2\theta_1)}} - 150 \\
&= 50\sqrt{\alpha^2-\sin^2\theta_1}\left(\frac{3}{\sin\theta_1} - \frac{1}{\sqrt{1-\sin^2\theta_1}}\right. \\
&\quad \left. + \frac{1}{\sqrt{1-\alpha^2+\sin^2\theta_1}}\right) - 150.
\end{aligned}$$

The final equation in terms of $u = \sin\theta_1$ then becomes

$$\sqrt{\alpha^2-u^2}\left(\frac{3}{u} - \frac{1}{\sqrt{1-u^2}} + \frac{1}{\sqrt{1-\alpha^2+u^2}}\right) = 3.$$

For $\alpha = 1.2$, this can be transformed into the following polynomial by squaring appropriately using Mathematica [20]:

$$\begin{aligned}
p(u) = &- 5602195930320001 + 93511401766200000u \\
&- 713160370741499900u^2 + 3259398736514250000u^3 \\
&- 9869397269940000000u^4 + 20717559301050000000u^5 \\
&- 30701172521250000000u^6 + 32082903984375000000u^7 \\
&- 23159988281250000000u^8 + 10999072265625000000u^9 \\
&- 3093750000000000000u^{10} + 390625000000000000u^{11}.
\end{aligned}$$

To show that polynomial $p(u)$ is unsolvable, we thus need three primes q_1, q_2, q_3 that adhere to the conditions in Lemma 6. Using Mathematica we find the following expressions for $p(u)$ modulo 59, 37, and 17, respectively:

$$\begin{aligned}
&46\big(u^{11} + 44u^{10} + 32u^9 + 33u^8 + 26u^7 + 47u^6 + 21u^5 \\
&\qquad + 11u^4 + 38u^3 + 3u^2 + 6u + 42\big),
\end{aligned}$$

$$\begin{aligned}
&16(u+17)\big(u^{10} + 18u^9 + 23u^8 + 23u^7 + 35u^6 + 8u^5 \\
&\qquad + 34u^4 + 16u^3 + 11u^2 + 34u + 10\big),
\end{aligned}$$

$$4\big(u^2+14u+9\big)\big(u^9+8u^8+11u^7+3u^6+5u^5 + 2u^4+2u^3+12u^2+9u+16\big).$$

We conclude that even the very limited weighted region problem where we allow for a single quadrant to have weight unequal to 1 and s and t are on halfplanes bounded by the sides of the quadrant, not containing the quadrant, is not solvable within the ACM$\mathbb{Q}$. □

4 Computing a Shortest Path Map

To find a shortest path from a source point s to *all* points at once, one can build a *Shortest Path Map* (*SPM*), see e.g., [10, 14, 15]. A *SPM* is a subdivision of the space for a given source s, where for each cell the paths $\pi(s,t)$, with t in the cell, have the same type. With it, we are able to find for each specific destination t, the weight of the shortest path from s to t simply by locating the point t in the subdivision. Once a *SPM* is available, we are able to report weights of shortest paths from s to any destination t by standard point location techniques [8, 12]. To compute the *SPM*, we consider computing the bisectors $b_{i,j} = \{q \mid q \in \mathbb{R}^2 \wedge d_i(s,q) = d_j(s,q)\}$ for all relevant pairs of shortest path types π_i, π_j, i.e., pairs for which $b_{i,j}$ appears in the Shortest Path Map. A *SPM* requires only polynomial space. However, in general, the bisector curves that bound cells of the *SPM* subdivision will be curves of very high degree [6, 11].

As before, we consider the setting where R is a rectangular region. In Section 4.1, we first consider the case when s lies on the boundary of R. In Section 4.2, we do the same for the case s lies inside R. The case that s lies outside R is not interesting, as we cannot even compute exactly a single shortest path in that case.

4.1 The source point s lies on the boundary of R

The *SPM* is given by the boundary of R and several bisector curves, expressed as points $(x, b_{i,j}(x))$. If $\alpha < 1$, these curves all lie outside R (the interior of R is a single region in the *SPM*).

Furthermore, bisectors involving $\pi_9(s,t)$ are of a much more complicated form, as might be expected from the implicit representation used for $d_9(s,t)$ in Theorem 3. Therefore, Lemmas 7 and 8 give the bisector curves, excluding the ones related to $\pi_9(s,t)$. The proofs are deferred to Appendix B.

Lemma 7 *The SPM for a point $s = (s_x, 0)$ on the boundary of the region R with weight $\alpha < 1$ is defined by:*

$$b_{i,j}(x) = \begin{cases} \frac{\sqrt{1-\alpha^2}}{\alpha}(s_x - x) & \text{if } i=1, j=2 \\ -\frac{\sqrt{1-\alpha^2}}{\alpha}x & \text{if } i=2, j=3 \\ 0 & \text{if } i=3, j=6 \end{cases}$$

Lemma 8 *The SPM for a point $s = (s_x, 0)$ on the boundary of the region R with weight $1 < \alpha < \sqrt{2}$ is defined by:*

$$b_{i,j}(x) = \begin{cases} 0 & \text{if } i=1, j=4 \\ \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}x & \text{if } i=4, j=5 \\ \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}x - \sqrt{\alpha^2-1}s_x & \text{if } i=5, j=6 \\ x = 0 & \text{if } i=6, j=7 \\ -1 - \frac{\sqrt{2-\alpha^2}}{\sqrt{\alpha^2-1}}x & \text{if } i=7, j=8 \\ -\sqrt{\alpha^2-1}(s_x - x) & \text{if } i=10, j=11 \\ -\frac{(s_x+x)+2\alpha\sqrt{s_x x}}{\sqrt{\alpha^2-1}} & \text{if } i=11, j=12 \end{cases}$$

We conjecture the following on the bisectors involving $\pi_9(s,t)$.

Conjecture 1 *No point on $b_{i,9}(x) \setminus R$, $i \in \{4, \ldots, 8\}$, can be computed exactly within ACM$\mathbb{Q}$.*

We tried to prove this conjecture by taking a similar approach as in Theorem 5. However, the solution to Equation (1) already seems to be of high degree. We therefore did not manage to formulate a point on the bisector as a polynomial equation (not containing roots).

Note that in the more restrictive case where R is a single quadrant and s lies on its boundary, the only types of shortest paths that exist are $\pi_i(s,t)$, for $i \in \{1,2,3,4,5,6,10,11,12\}$. Thus, we can compute the *SPM* in the ACM$\mathbb{Q}$ (the bisectors are given by some of the equations in Lemmas 7 and 8).

4.2 The source point s lies inside R

In this case we have shortest paths of type $\pi_i(s,t)$, for $i \in \{6,7,8,9,11,12\}$. Hence, the equations of the bisectors of the *SPM* are given by the sides of R, and bisector $b_{6,9}$ if $\alpha < 1$, and bisectors $b_{6,7}, b_{7,8}, b_{6,9}, b_{7,9}, b_{8,9}$ and $b_{11,12}$ if $1 < \alpha < \sqrt{2}$. See Lemmas 7 and 8, and Conjecture 1.

5 Conclusion

We analyzed the WRP when there is only one weighted rectangle R, and showed how to obtain the exact shortest path $\pi(s,t)$ and its length when s lies in or on R. When both s and t lie outside R the exact solution is unsolvable in the ACM$\mathbb{Q}$. We obtain similar results in the case where R is a single quadrant. For future work, it would be interesting to find an exact formula within the ACM$\mathbb{Q}$ for the bisectors involving $\pi_9(s,t)$. In addition, we may want to analyze if or how we can generalize these results to other convex shapes.

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. Approximation algorithms for geometric shortest path problems. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 286–295, 2000.

[3] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM*, 52(1):25–53, 2005.

[4] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3:177–191, 1988.

[5] P. Bose, G. Esteban, and A. Maheshwari. A Steiner-point-based algorithm for approximate shortest paths in weighted equilateral-triangle meshes. *Theoretical Computer Science*, page 114583, 2024.

[6] J.-L. De Carufel, C. Grimm, A. Maheshwari, M. Owen, and M. H. M. Smid. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry*, 47(7):724–727, 2014.

[7] D. S. Dummit and R. M. Foote. *Abstract Algebra*, volume 3. Wiley Hoboken, 2004.

[8] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

[9] L. Gewali, A. Meng, J. S. B. Mitchell, and S. Ntafos. Path planning in O/1/infinity weighted regions with applications. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[10] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[11] J. Hershberger, S. Suri, and H. Yildiz. A near-optimal algorithm for shortest paths among curved obstacles in the plane. *SIAM Journal on Computing*, 51(4):1296–1340, 2022.

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

[13] J. S. B. Mitchell. Shortest paths among obstacles, zero-cost regions, and roads. Technical report, Cornell University Operations Research and Industrial Engineering, 1987.

[14] J. S. B. Mitchell. Shortest paths among obstacles in the plane. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, pages 308–317, 1993.

[15] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[16] S. Narayanappa. *Geometric routing*. PhD thesis, University of Denver, 2006.

[17] G. J. E. Rawlins and D. Wood. Ortho-convexity and its generalizations. In *Machine Intelligence and Pattern Recognition*, volume 6, pages 137–152. Elsevier, 1988.

[18] Z. Sun and J. H. Reif. On finding approximate optimal paths in weighted regions. *Journal of Algorithms*, 58(1):1–32, 2006.

[19] H. Wang. A new algorithm for Euclidean shortest paths in the plane. *Journal of the ACM*, 70(2):1–62, 2023.

[20] Wolfram Research, Inc. Mathematica, Version 13.3. Champaign, IL, 2023.

A Computing a shortest path when the source point s lies on the boundary of R

In this section, we prove the stated lengths of the shortest paths as defined in Theorems 3 and 4. All angles in this section are measured relative to the vertical or horizontal line through the bending point for the bending point on the top (and bottom) or left side of R, respectively. We denote by θ_c the critical angle. The following two equations capture the properties we use of the critical angle. For $\alpha < 1$, we have

$$\sin\theta_c = \alpha \Rightarrow \begin{cases} \cos\theta_c = \sqrt{1-\alpha^2} \\ \tan\theta_c = \frac{\sin\theta_c}{\cos\theta_c} = \frac{\alpha}{\sqrt{1-\alpha^2}}. \end{cases} \tag{4}$$

And for $\alpha > 1$, we have

$$\sin\theta_c = \frac{1}{\alpha} \Rightarrow \begin{cases} \cos\theta_c = \sqrt{1-\frac{1}{\alpha^2}} \\ \tan\theta_c = \frac{\sin\theta_c}{\cos\theta_c} = \frac{\frac{1}{\alpha}}{\sqrt{1-\frac{1}{\alpha^2}}} = \frac{1}{\sqrt{\alpha^2-1}}. \end{cases} \tag{5}$$

We frequently use these equations in the rest of this section to determine the lengths of the path $d_i(s,t)$ for all i.

From now on, we consider the case where the source point s is restricted to the boundary of R, an axis-aligned rectangle of unit height with top-left corner at $(0,0)$.

Observation 3 *Let R be a rectangular region with weight $0 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest path $\pi_1(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_1(s,t) = \sqrt{(s_x - t_x)^2 + t_y^2}$.*

Lemma 9 *Let R be a rectangular region with weight $0 < \alpha < 1$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest paths $\pi_2(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_2(s,t) = \alpha(s_x - t_x) + \sqrt{1-\alpha^2}t_y$.*

Proof. Let θ_c be the critical angle made by $\pi_2(s,t)$ on the top boundary of R, and let $(b,0)$ be the point where the shortest path leaves R. We use Equation (4) to obtain the value of b:

$$\frac{\alpha}{\sqrt{1-\alpha^2}} = \frac{|b-t_x|}{|t_y|} = \frac{b-t_x}{t_y} \Rightarrow b = t_x + \frac{\alpha}{\sqrt{1-\alpha^2}}t_y. \tag{6}$$

We know that the weight of the shortest paths $\pi_2(s,t)$ is given by $d_2(s,t) = \alpha|s_x - b| + \sqrt{(b-t_x)^2 + t_y^2}$. By using Equation (6), we have that

$$\begin{aligned} d_2(s,t) &= \alpha\left(s_x - t_x - \frac{\alpha}{\sqrt{1-\alpha^2}}t_y\right) \\ &\quad + \sqrt{\left(t_x + \frac{\alpha}{\sqrt{1-\alpha^2}}t_y - t_x\right)^2 + t_y^2} \\ &= \alpha\left(s_x - t_x - \frac{\alpha}{\sqrt{1-\alpha^2}}t_y\right) + \sqrt{\frac{\alpha^2 t_y^2}{1-\alpha^2} + t_y^2} \\ &= \alpha(s_x - t_x) - \frac{\alpha^2 t_y}{\sqrt{1-\alpha^2}} + \sqrt{\frac{t_y^2}{1-\alpha^2}} \\ &= \alpha(s_x - t_x) - \frac{\alpha^2 t_y}{\sqrt{1-\alpha^2}} + \frac{|t_y|}{\sqrt{1-\alpha^2}} \\ &= \alpha(s_x - t_x) + \frac{1-\alpha^2}{\sqrt{1-\alpha^2}}t_y \\ &= \alpha(s_x - t_x) + \sqrt{1-\alpha^2}t_y. \end{aligned}$$

□

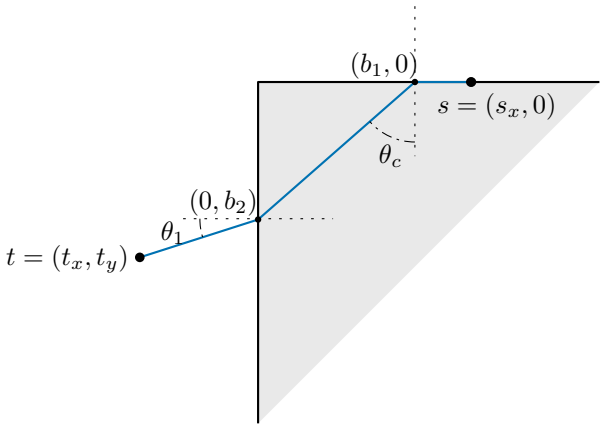


Figure 4: Illustration of Lemma 10.

Observation 4 *Let R be a rectangular region with weight $0 < \alpha < 1$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest paths $\pi_3(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_3(s,t) = \alpha s_x + \sqrt{t_x^2 + t_y^2}$.*

Observation 5 *Let R be a rectangular region with weight $1 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest path $\pi_4(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_4(s,t) = s_x + \sqrt{t_x^2 + t_y^2}$.*

Lemma 10 *Let R be a rectangular region with weight $1 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest path $\pi_5(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_5(s,t) = s_x - \sqrt{2-\alpha^2}t_x - \sqrt{\alpha^2 - 1}t_y$.*

Proof. The shortest path from s to t intersects the top side of R, and then it enters R using the critical angle, see Figure 4. We proceed to compute the coordinates of the vertices of the shortest path in this case.

Let θ_1 be the angle at which the shortest path leaves R with respect to the normal, see Figure 4. Then

$$\sin\theta_1 = \alpha\sin\left(\frac{\pi}{2} - \theta_c\right) = \alpha\cos\theta_c = \sqrt{\alpha^2 - 1},$$

and thus

$$\tan\theta_1 = \frac{\sin\theta_1}{\cos\theta_1} = \frac{\sqrt{\alpha^2-1}}{\sqrt{1-(\alpha^2-1)}} = \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}. \tag{7}$$

Let $(b_1, 0)$ and $(0, b_2)$ be, respectively, the points where the shortest path enters and leaves R. We also know that $\tan\theta_1 = \frac{|t_y - b_2|}{|t_x|}$. Since $t_y < b_2 < 0$, and $t_x < 0$, we use Equation (7) to get the value of b_2:

$$\begin{aligned} \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}} = \frac{|t_y - b_2|}{|t_x|} &= \frac{b_2 - t_y}{-t_x} \\ \Rightarrow b_2 &= t_y - \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}t_x. \end{aligned} \tag{8}$$

Also, since $\tan\theta_c = \frac{|b_1|}{|b_2|}$, and $b_2 < 0 < b_1$, we use Equation (5) to get the value of b_1:

$$\frac{1}{\sqrt{\alpha^2-1}} = \frac{|b_1|}{|b_2|} = \frac{b_1}{-b_2}$$

$$\Rightarrow b_1 = -\frac{b_2}{\sqrt{\alpha^2-1}} = -\frac{t_y}{\sqrt{\alpha^2-1}} + \frac{t_x}{\sqrt{2-\alpha^2}}.$$

Since $\sin\theta_c = \frac{b_1}{\sqrt{b_1^2+b_2^2}} = \frac{1}{\alpha} \Rightarrow \sqrt{b_1^2+b_2^2} = b_1\alpha$. Thus:

$$\begin{aligned}
d_5(s,t) &= s_x - b_1 + \alpha\sqrt{b_1^2+b_2^2} + \sqrt{t_x^2+(t_y-b_2)^2} \\
&= s_x - b_1 + b_1\alpha^2 + \sqrt{t_x^2 + \left(t_y - t_y + \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}t_x\right)^2} \\
&= s_x + (\alpha^2-1)b_1 + \sqrt{t_x^2 + \frac{\alpha^2-1}{2-\alpha^2}t_x^2} \\
&= s_x + (\alpha^2-1)b_1 + |t_x|\sqrt{1+\frac{\alpha^2-1}{2-\alpha^2}} \\
&= s_x + (\alpha^2-1)b_1 + |t_x|\sqrt{\frac{2-\alpha^2+\alpha^2-1}{2-\alpha^2}} \\
&= s_x + (\alpha^2-1)\left(\frac{t_x}{\sqrt{2-\alpha^2}} - \frac{t_y}{\sqrt{\alpha^2-1}}\right) - t_x\frac{1}{\sqrt{2-\alpha^2}} \\
&= s_x - \sqrt{\alpha^2-1}t_y + \frac{\alpha^2-1-1}{\sqrt{2-\alpha^2}}t_x \\
&= s_x - \sqrt{\alpha^2-1}t_y - \frac{2-\alpha^2}{\sqrt{2-\alpha^2}}t_x \\
&= s_x - \sqrt{\alpha^2-1}t_y - \sqrt{2-\alpha^2}t_x. \qquad \square
\end{aligned}$$

Lemma 11 *Let R be a rectangular region with weight $0 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest paths $\pi_6(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_6(s,t) = \alpha\sqrt{s_x^2+y^2} + \sqrt{t_x^2+(t_y-y)^2}$, where y is the unique real solution in the interval $(t_y, 0)$ to the equation*

$$\begin{aligned}
(\alpha^2-1)y^4 - 2t_y(\alpha^2-1)y^3 + [\alpha^2t_x^2 + (\alpha^2-1)t_y^2 - s_x^2]y^2 \\
+2s_x^2t_y y - s_x^2t_y^2 = 0,
\end{aligned}$$

Proof. Let $(0,y)$ be the point where $\pi_6(s,t)$ leaves R, and let θ_1 and θ_2 be, respectively, the angles of incidence and refraction at $(0,y)$. Then, by Snell's law of refraction, we get that $\alpha\sin\theta_1 = \sin\theta_2$. Thus,

$$\begin{aligned}
\alpha\frac{|y|}{\sqrt{s_x^2+y^2}} &= \frac{|t_y-y|}{\sqrt{t_x^2+(t_y-y)^2}} \\
\Rightarrow \alpha^2y^2(t_x^2+(t_y-y)^2) &= (t_y-y)^2(s_x^2+y^2) \\
\Rightarrow \alpha^2y^2t_x^2 + \alpha^2y^2t_y^2 + \alpha^2y^4 - 2\alpha^2y^3t_y &= s_x^2t_y^2 + s_x^2y^2 - 2s_x^2t_y y \\
&\quad + y^2t_y^2 + y^4 - 2y^3t_y.
\end{aligned}$$

Hence,

$$\begin{aligned}
(\alpha^2-1)y^4 - 2t_y(\alpha^2-1)y^3 + [\alpha^2t_x^2 + (\alpha^2-1)t_y^2 - s_x^2]y^2 \\
+2s_x^2t_y y - s_x^2t_y^2 = 0
\end{aligned}$$

Finally, we get that the weighted length of the shortest paths $\pi_6(s,t)$ is given by

$$d_6(s,t) = \alpha\sqrt{s_x^2+y^2} + \sqrt{t_x^2+(t_y-y)^2}. \qquad \square$$

Lemma 12 *Let R be a rectangular region with weight $1 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest paths $\pi_7(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_7(s,t) = \sqrt{\alpha^2-1}s_x + 1 + \sqrt{t_x^2+(t_y+1)^2}$.*

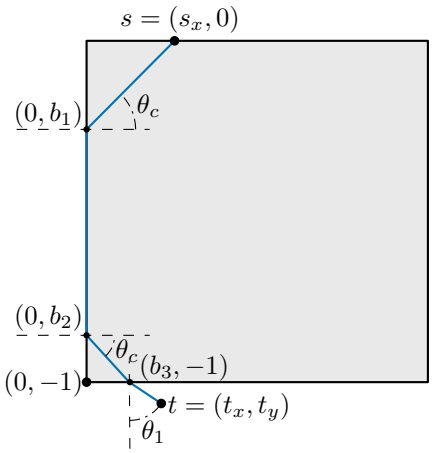


Figure 5: Illustration of Lemma 13.

Proof. Let $(0, b_1)$ be the point where $\pi_7(s,t)$ leaves R, i.e., the first vertex of the shortest path. Since $b_1 < 0$, and $s_x > 0$, we obtain the coordinates of this first bending point by using Equation (5):

$$\tan\theta_c = \frac{|b_1|}{|s_x|} = \frac{-b_1}{s_x} = \frac{1}{\sqrt{\alpha^2-1}} \Rightarrow b_1 = -\frac{s_x}{\sqrt{\alpha^2-1}}. \quad (9)$$

The weight of the shortest paths $\pi_7(s,t)$ is then given by

$$\begin{aligned}
d_7(s,t) &= \alpha\sqrt{s_x^2+b_1^2} + (b_1+1) + \sqrt{t_x^2+(-1-t_y)^2} \\
&= \frac{\alpha^2 s_x}{\sqrt{\alpha^2-1}} - \frac{s_x}{\sqrt{\alpha^2-1}} + 1 + \sqrt{t_x^2+(-1-t_y)^2} \\
&= \sqrt{\alpha^2-1}s_x + 1 + \sqrt{t_x^2+(t_y+1)^2}. \qquad \square
\end{aligned}$$

Lemma 13 *Let R be a rectangular region with weight $1 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest path $\pi_8(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_8(s,t) = \sqrt{\alpha^2-1}(s_x + t_x) - \sqrt{2-\alpha^2}(1+t_y) + 1$.*

Proof. Let $(0, b_1)$ be the point where $\pi_8(s,t)$ leaves R for the first time and let $(0, b_2)$ and $(b_3, -1)$ be, respectively, the points where $\pi_8(s,t)$ enters and leaves R for the second time, see Figure 5. As $\pi_7(s,t)$ and $\pi_8(s,t)$ overlap up to b_2, Equation (9) gives us that $b_1 = -\frac{s_x}{\sqrt{\alpha^2-1}}$.

Recall that R has height 1. Let θ_1 be the angle at which the shortest path leaves R for the last time with respect to the normal, see Figure 5. Then, $\tan\theta_1 = \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}$, similar to Equation (7). So,

$$\begin{aligned}
\tan\theta_1 = \frac{|t_x-b_3|}{|t_y+1|} = \frac{t_x-b_3}{-1-t_y} &= \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}} \\
\Rightarrow b_3 = t_x + (1+t_y)\frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}&.
\end{aligned}$$

And, using Equation (5), we get

$$\tan\theta_c = \frac{|-1-b_2|}{|b_3|} = \frac{b_2+1}{b_3} = \frac{1}{\sqrt{\alpha^2-1}}$$

$$\Rightarrow b_2 = \frac{b_3}{\sqrt{\alpha^2-1}} - 1 = \frac{t_x}{\sqrt{\alpha^2-1}} + \frac{1+t_y}{\sqrt{2-\alpha^2}} - 1.$$

The weight of the shortest paths $\pi_8(s,t)$ is given by $d_8(s,t) = \alpha\sqrt{s_x^2+b_1^2} + |b_2 - b_1| + \alpha\sqrt{b_3^2+(b_2+1)^2} + \sqrt{(b_3-t_x)^2+(-1-t_y)^2}$. Using the expression for b_1, we have that

$$\sqrt{s_x^2+b_1^2} = \sqrt{s_x^2 + \left(-\frac{s_x}{\sqrt{\alpha^2-1}}\right)^2} = \sqrt{\frac{\alpha^2-1+1}{\alpha^2-1}s_x^2} = \frac{\alpha s_x}{\sqrt{\alpha^2-1}}. \quad (10)$$

Using our expressions for b_2 and b_3, and the fact that $b_2+1 > 0$, we obtain the following for the terms $A = \sqrt{b_3^2+(b_2+1)^2}$ and $B = \sqrt{(b_3-t_x)^2+(-1-t_y)^2}$:

$$\begin{aligned}
A &= \sqrt{\left(t_x + (1+t_y)\frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}\right)^2 + \left(\frac{t_x}{\sqrt{\alpha^2-1}} + \frac{1+t_y}{\sqrt{2-\alpha^2}} - 1 + 1\right)^2} \\
&= \sqrt{(\alpha^2-1)\left(\frac{t_x}{\sqrt{\alpha^2-1}} + \frac{1+t_y}{\sqrt{2-\alpha^2}}\right)^2 + \left(\frac{t_x}{\sqrt{\alpha^2-1}} + \frac{1+t_y}{\sqrt{2-\alpha^2}}\right)^2} \\
&= \sqrt{\alpha^2\left(\frac{t_x}{\sqrt{\alpha^2-1}} + \frac{1+t_y}{\sqrt{2-\alpha^2}}\right)^2} \\
&= \alpha\left(\frac{t_x}{\sqrt{\alpha^2-1}} + \frac{1+t_y}{\sqrt{2-\alpha^2}}\right), \quad (11)
\end{aligned}$$

and

$$\begin{aligned}
B &= \sqrt{\left(t_x + (1+t_y)\frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}} - t_x\right)^2 + (-1-t_y)^2} \\
&= \sqrt{(1+t_y)^2\frac{\alpha^2-1}{2-\alpha^2} + (1+t_y)^2} \\
&= \sqrt{(1+t_y)^2\frac{\alpha^2-1+2-\alpha^2}{2-\alpha^2}} \\
&= \frac{|1+t_y|}{\sqrt{2-\alpha^2}}. \quad (12)
\end{aligned}$$

Using Equations (10), (11), and (12), we get that the weighted length of the shortest paths $\pi_8(s,t)$ is given by

$$\begin{aligned}
d_8(s,t) &= \frac{\alpha^2 s_x}{\sqrt{\alpha^2-1}} - \frac{s_x+t_x}{\sqrt{\alpha^2-1}} - \frac{1+t_y}{\sqrt{2-\alpha^2}} + 1 \\
&\quad + \alpha^2\left(\frac{t_x}{\sqrt{\alpha^2-1}} + \frac{1+t_y}{\sqrt{2-\alpha^2}}\right) + \frac{|1+t_y|}{\sqrt{2-\alpha^2}} \\
&= \alpha^2\frac{s_x+t_x}{\sqrt{\alpha^2-1}} - \frac{s_x+t_x}{\sqrt{\alpha^2-1}} + (\alpha^2-1)\frac{1+t_y}{\sqrt{2-\alpha^2}} \\
&\quad + 1 - \frac{1+t_y}{\sqrt{2-\alpha^2}} \\
&= (\alpha^2-1)\frac{s_x+t_x}{\sqrt{\alpha^2-1}} + (\alpha^2-2)\frac{1+t_y}{\sqrt{2-\alpha^2}} + 1 \\
&= \sqrt{\alpha^2-1}(s_x+t_x) - \sqrt{2-\alpha^2}(1+t_y) + 1. \quad \square
\end{aligned}$$

Lemma 14 *Let R be a rectangular region with weight $0 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest paths $\pi_9(s,t)$ from s to a point $t = (t_x, t_y)$ outside R is given by $d_9(s,t) = \alpha\sqrt{(s_x-x)^2+1} + \sqrt{(t_x-x)^2+(t_y+1)^2}$ where x is the unique real solution in the interval (t_x, s_x) to the equation*

$$\begin{aligned}
&(\alpha^2-1)x^4 - 2(\alpha^2-1)(t_x+s_x)x^3 \\
&+ [(\alpha^2-1)(s_x^2+t_x^2+4s_xt_x) + \alpha^2(1+t_y)^2 - 1]x^2 \\
&- 2[(\alpha^2-1)(t_xs_x^2+t_x^2s_x) + \alpha^2(1+t_y)^2s_x - t_x]x \\
&+ (\alpha^2-1)t_x^2s_x^2 + \alpha^2(1+t_y)^2s_x^2 - t_x^2 = 0.
\end{aligned}$$

Proof. Let $(x,-1)$ be the point where $\pi_9(s,t)$ leaves R, and let θ_1 and θ_2 be, respectively, the angles of incidence and refraction at $(x,-1)$. Then, by Snell's law of refraction, we get that:

$$\begin{aligned}
&\alpha\sin\theta_1 = \sin\theta_2 \Rightarrow \alpha\frac{s_x-x}{\sqrt{(s_x-x)^2+1}} = \frac{x-t_x}{\sqrt{(x-t_x)^2+(-1-t_y)^2}} \\
&\Rightarrow \alpha^2(x-t_x)^2(s_x-x)^2 + \alpha^2(-1-t_y)^2(s_x-x)^2 \\
&\quad = (x-t_x)^2(s_x-x)^2 + (x-t_x)^2 \\
&\Rightarrow (\alpha^2-1)(x-t_x)^2(s_x-x)^2 + \alpha^2(-1-t_y)^2(s_x-x)^2 \\
&\quad - (x-t_x)^2 = 0 \\
&\Rightarrow \left[(\alpha^2-1)x^2 + (\alpha^2-1)t_x^2 - 2(\alpha^2-1)t_xx\right] \\
&\quad \cdot (s_x^2+x^2-2s_xx) + \alpha^2(-1-t_y)^2s_x^2 + \alpha^2(-1-t_y)^2x^2 \\
&\quad - 2\alpha^2(-1-t_y)^2s_xx - x^2 - t_x^2 + 2t_xx = 0 \\
&\Rightarrow (\alpha^2-1)x^2s_x^2 + (\alpha^2-1)x^4 - 2(\alpha^2-1)s_xx^3 \\
&\quad + (\alpha^2-1)t_x^2s_x^2 + (\alpha^2-1)t_x^2x^2 - 2(\alpha^2-1)t_x^2s_xx \\
&\quad - 2(\alpha^2-1)t_xs_x^2x - 2(\alpha^2-1)t_xx^3 + 4(\alpha^2-1)s_xt_xx^2 \\
&\quad + \alpha^2(-1-t_y)^2s_x^2 + \alpha^2(-1-t_y)^2x^2 \\
&\quad - 2\alpha^2(-1-t_y)^2s_xx - x^2 - t_x^2 + 2t_xx = 0.
\end{aligned}$$

Hence,

$$\begin{aligned}
&(\alpha^2-1)x^4 - 2(\alpha^2-1)(t_x+s_x)x^3 \\
&+ [(\alpha^2-1)(s_x^2+t_x^2+4s_xt_x) + \alpha^2(1+t_y)^2 - 1]x^2 \\
&- 2[(\alpha^2-1)(t_xs_x^2+t_x^2s_x) + \alpha^2(1+t_y)^2s_x - t_x]x \\
&+ (\alpha^2-1)t_x^2s_x^2 + \alpha^2(1+t_y)^2s_x^2 - t_x^2 = 0.
\end{aligned}$$

Finally, we get that the weighted length of the shortest paths $\pi_9(s,t)$ is given by

$$d_9(s,t) = \alpha\sqrt{(s_x-x)^2+1} + \sqrt{(t_x-x)^2+(1+t_y)^2}. \quad \square$$

Lemma 15 *Let R be a rectangular region with weight $1 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest paths $\pi_{10}(s,t)$ from s to a point $t = (t_x, t_y)$ inside R is given by $d_{10}(s,t) = s_x - t_x - \sqrt{\alpha^2-1}t_y$.*

Proof. Let $(b_1, 0)$ be the point where $\pi_{10}(s,t)$ enters R. Let θ_c be the angle at which $\pi_{10}(s,t)$ enters R. Since θ_c is the critical angle, using Equation (5), we get the value of b_1:

$$\begin{aligned}
\tan\theta_c = \frac{|b_1-t_x|}{|t_y|} = \frac{b_1-t_x}{-t_y} &= \frac{1}{\sqrt{\alpha^2-1}} \\
\Rightarrow b_1 = t_x - \frac{t_y}{\sqrt{\alpha^2-1}}. &\quad (13)
\end{aligned}$$

We know that the weight of the shortest paths $\pi_{10}(s,t)$ is given by $d_{10}(s,t) = |s_x - b_1| + \alpha\sqrt{(b_1 - t_x)^2 + t_y^2}$. By using Equation (13), we have that

$$\begin{aligned}
d_{10}(s,t) &= \left(s_x - t_x + \frac{t_y}{\sqrt{\alpha^2-1}}\right) \\
&\quad + \alpha\sqrt{\left(t_x - \frac{t_y}{\sqrt{\alpha^2-1}} - t_x\right)^2 + t_y^2} \\
&= \left(s_x - t_x + \frac{t_y}{\sqrt{\alpha^2-1}}\right) + \alpha\sqrt{\frac{t_y^2}{\alpha^2-1} + t_y^2} \\
&= s_x - t_x + \frac{t_y}{\sqrt{\alpha^2-1}} + \alpha\sqrt{\frac{\alpha^2 t_y^2}{\alpha^2-1}} \\
&= s_x - t_x + \frac{t_y}{\sqrt{\alpha^2-1}} + \frac{\alpha^2|t_y|}{\sqrt{\alpha^2-1}} \\
&= s_x - t_x + \frac{t_y}{\sqrt{\alpha^2-1}} - \frac{\alpha^2 t_y}{\sqrt{\alpha^2-1}} \\
&= s_x - t_x - \frac{(\alpha^2-1)t_y}{\sqrt{\alpha^2-1}} \\
&= s_x - t_x - \sqrt{\alpha^2-1}t_y. \qquad \square
\end{aligned}$$

Observation 6 *Let R be a rectangular region with weight $0 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest paths $\pi_{11}(s,t)$ from s to a point $t = (t_x, t_y)$ inside R is given by $d_{11}(s,t) = \alpha\sqrt{(s_x - t_x)^2 + t_y^2}$.*

Lemma 16 *Let R be a rectangular region with weight $1 < \alpha < \sqrt{2}$. Let $s = (s_x, 0)$ be a point on the boundary of R. Then the length of the shortest path $\pi_{12}(s,t)$ from s to a point $t = (t_x, t_y)$ inside R is given by $d_{12}(s,t) = \sqrt{\alpha^2-1}(s_x + t_x) - t_y$.*

Proof. Let $(0, b_1)$ and $(0, b_2)$ be the points where $\pi_{12}(s,t)$ leaves and enters for the second time, respectively, the region R. From Lemma 13 we know that $b_1 = -\frac{s_x}{\sqrt{\alpha^2-1}}$. Using Equation (5), we find:

$$\begin{aligned}
\tan\theta_c = \frac{|t_y - b_2|}{|t_x|} = \frac{b_2 - t_y}{t_x} &= \frac{1}{\sqrt{\alpha^2-1}} \\
\Rightarrow b_2 &= \frac{t_x}{\sqrt{\alpha^2-1}} + t_y.
\end{aligned}$$

We then get that the weight of $\pi_{12}(s,t)$ is given by $d_{12}(s,t) = \alpha\sqrt{s_x^2 + b_1^2} + |b_2 - b_1| + \alpha\sqrt{t_x^2 + (b_2 - t_y)^2}$. So:

$$\begin{aligned}
d_{12}(s,t) &= \alpha\sqrt{s_x^2 + \frac{s_x^2}{\alpha^2-1}} - \frac{s_x}{\sqrt{\alpha^2-1}} - \frac{t_x}{\sqrt{\alpha^2-1}} - t_y \\
&\quad + \alpha\sqrt{t_x^2 + \left(\frac{t_x}{\sqrt{\alpha^2-1}} + t_y - t_y\right)^2} \\
&= \alpha\sqrt{\frac{\alpha^2 s_x^2}{\alpha^2-1}} - \frac{s_x + t_x}{\sqrt{\alpha^2-1}} - t_y + \alpha\sqrt{t_x^2 + \frac{t_x^2}{\alpha^2-1}} \\
&= \frac{\alpha^2|s_x|}{\sqrt{\alpha^2-1}} - \frac{s_x + t_x}{\sqrt{\alpha^2-1}} - t_y + \alpha\sqrt{\frac{\alpha^2 t_x^2}{\alpha^2-1}} \\
&= \frac{\alpha^2 s_x}{\sqrt{\alpha^2-1}} - \frac{s_x + t_x}{\sqrt{\alpha^2-1}} - t_y + \frac{\alpha^2 t_x}{\sqrt{\alpha^2-1}} \\
&= \frac{\alpha^2(s_x + t_x)}{\sqrt{\alpha^2-1}} - \frac{s_x + t_x}{\sqrt{\alpha^2-1}} - t_y \\
&= \frac{\alpha^2-1}{\sqrt{\alpha^2-1}}(s_x + t_x) - t_y \\
&= \sqrt{\alpha^2-1}(s_x + t_x) - t_y. \qquad \square
\end{aligned}$$

B Computing the shortest path map for a given s

In this section, we express each bisector as an explicit function of the shape $y = f(x)$. Then, the actual bisector is given by the corresponding points (x, y).

Lemma 17 *The bisector $b_{1,2}$ is given by $y = \frac{\sqrt{1-\alpha^2}}{\alpha}(s_x - x)$.*

Proof. We want to compute the coordinates of the points such that the weighted length of paths $\pi_1(s,t)$ and $\pi_2(s,t)$ is the same, i.e., the points (t_x, t_y) such that $\sqrt{(s_x - t_x)^2 + t_y^2} = \alpha(s_x - t_x) + \sqrt{1-\alpha^2}t_y$. Thus:

$$\begin{aligned}
(s_x - t_x)^2 + t_y^2 &= \alpha^2(s_x - t_x)^2 + (1-\alpha^2)t_y^2 \\
&\quad + 2\alpha\sqrt{1-\alpha^2}(s_x - t_x)t_y \\
0 &= (1-\alpha^2)(s_x - t_x)^2 + \alpha^2 t_y^2 \\
&\quad - 2\alpha\sqrt{1-\alpha^2}(s_x - t_x)t_y \\
0 &= \left[\sqrt{1-\alpha^2}(s_x - t_x) - \alpha t_y\right]^2 \\
\alpha t_y &= \sqrt{1-\alpha^2}(s_x - t_x) \\
\Rightarrow t_y &= \frac{\sqrt{1-\alpha^2}}{\alpha}(s_x - t_x). \qquad \square
\end{aligned}$$

Lemma 18 *The bisector $b_{2,3}$ is given by $y = -\frac{\sqrt{1-\alpha^2}}{\alpha}x$.*

Proof. We want to compute the coordinates of the points such that the weighted length of paths $\pi_2(s,t)$ and $\pi_3(s,t)$ is the same, i.e., the points (t_x, t_y) such that $\alpha(s_x - t_x) + \sqrt{1-\alpha^2}t_y = \alpha s_x + \sqrt{t_x^2 + t_y^2}$. Thus:

$$\begin{aligned}
-\alpha t_x + \sqrt{1-\alpha^2}t_y &= \sqrt{t_x^2 + t_y^2} \\
\alpha^2 t_x^2 + (1-\alpha^2)t_y^2 - 2\alpha\sqrt{1-\alpha^2}t_x t_y &= t_x^2 + t_y^2 \\
(1-\alpha^2)t_x^2 + \alpha^2 t_y^2 + 2\alpha\sqrt{1-\alpha^2}t_x t_y &= 0 \\
\left[\sqrt{1-\alpha^2}t_x + \alpha t_y\right]^2 &= 0 \\
\sqrt{1-\alpha^2}t_x &= -\alpha t_y \\
\Rightarrow t_y &= -\frac{\sqrt{1-\alpha^2}}{\alpha}t_x. \qquad \square
\end{aligned}$$

Lemma 19 *The bisector $b_{4,5}$ is given by $y = \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}x$.*

Proof. We want to compute the coordinates of the points such that the weighted length of paths $\pi_4(s,t)$ and $\pi_5(s,t)$ is the same, i.e., the points (t_x, t_y) such that $s_x + \sqrt{t_x^2 + t_y^2} = s_x - \sqrt{2-\alpha^2}t_x - \sqrt{\alpha^2-1}t_y$. Thus:

$$\begin{aligned}
\sqrt{t_x^2 + t_y^2} &= -\sqrt{\alpha^2-1}t_y - \sqrt{2-\alpha^2}t_x \\
t_x^2 + t_y^2 &= (\alpha^2-1)t_y^2 + (2-\alpha^2)t_x^2 + 2\sqrt{\alpha^2-1}\sqrt{2-\alpha^2}t_x t_y
\end{aligned}$$

$$
\begin{aligned}
0 &= (2-\alpha^2)t_y^2 - 2\sqrt{\alpha^2-1}\sqrt{2-\alpha^2}t_x t_y + (\alpha^2-1)t_x^2 \\
0 &= (\sqrt{2-\alpha^2}t_y - \sqrt{\alpha^2-1}t_x)^2 \\
\sqrt{2-\alpha^2}t_y &= \sqrt{\alpha^2-1}t_x \\
\Rightarrow t_y &= \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}t_x.
\end{aligned}
$$

□

Lemma 20 *The bisector $b_{5,6}$ is given by $y = \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}x - \sqrt{\alpha^2-1}s_x$.*

Proof. The path $\pi_6(s,t)$ is similar to a path $\pi_5(s,t)$ where the point of entry in R is s. The path $\pi_6(s,t)$ do not have the critical angle property. However, the points on the bisector, still have that critical angle property, since the shortest path from s to them is of both types. Let $(0,b_2)$ be the point where $\pi_5(s,t)$ leaves the square, see Figure 4. Using Equation (5) we obtain the following relation:

$$
\begin{aligned}
\tan\theta_c &= \frac{|s_x|}{|b_2|} \\
\Rightarrow |b_2| = \frac{|s_x|}{\tan\theta_c} &= \sqrt{\alpha^2-1}|s_x| = \sqrt{\alpha^2-1}s_x \\
\Rightarrow b_2 &= -\sqrt{\alpha^2-1}s_x.
\end{aligned}
$$

For $\pi_5(s,t)$, we obtained $b_2 = t_y - \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}t_x$ in Lemma 10 (see Equation (8)). We then obtain the equation of the bisector $b_{5,6}$:

$$
t_y = b_2 + \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}t_x = \frac{\sqrt{\alpha^2-1}}{\sqrt{2-\alpha^2}}t_x - \sqrt{\alpha^2-1}s_x.
$$

□

Lemma 21 *The bisector $b_{7,8}$ is given by $y = -1 - \frac{\sqrt{2-\alpha^2}}{\sqrt{\alpha^2-1}}x$.*

Proof. We want to know the coordinates of the points (t_x,t_y) such that $\sqrt{\alpha^2-1}(s_x+t_x) - \sqrt{2-\alpha^2}(1+t_y) + 1 = \sqrt{\alpha^2-1}s_x + 1 + \sqrt{t_x^2+(t_y+1)^2}$. Thus,

$$
\begin{aligned}
&\sqrt{\alpha^2-1}t_x - \sqrt{2-\alpha^2}(1+t_y) = \sqrt{t_x^2+(1+t_y)^2} \\
\Rightarrow\ &(\alpha^2-1)t_x^2 + (2-\alpha^2)(1+t_y)^2 \\
&-2\sqrt{\alpha^2-1}\sqrt{2-\alpha^2}(1+t_y)t_x = t_x^2 + (1+t_y)^2 \\
\Rightarrow\ &(2-\alpha^2)t_x^2 + (\alpha^2-1)(1+t_y)^2 \\
&+2\sqrt{2-\alpha^2}\sqrt{\alpha^2-1}(1+t_y)t_x = 0 \\
\Rightarrow\ &\left[\sqrt{2-\alpha^2}t_x + \sqrt{\alpha^2-1}(1+t_y)\right]^2 = 0 \\
&\Rightarrow -\sqrt{\alpha^2-1}(1+t_y) = \sqrt{2-\alpha^2}t_x \\
&\Rightarrow -1 - \frac{\sqrt{2-\alpha^2}}{\sqrt{\alpha^2-1}}t_x = t_y.
\end{aligned}
$$

□

Lemma 22 *The bisector $b_{10,11}$ is given by $y = -\sqrt{\alpha^2-1}(s_x - x)$.*

Proof. We want the curve defining the bisector between the region containing the points $t = (t_x,t_y)$ such that the shortest path from $s = (s_x,0)$ to t is $\pi_{10}(s,t)$, and the region containing the points $t = (t_x,t_y)$ such that the shortest path from $s = (s_x,0)$ to t is $\pi_{11}(s,t)$. Thus,

$$
\begin{aligned}
&\alpha\sqrt{(s_x-t_x)^2+t_y^2} = s_x - t_x - \sqrt{\alpha^2-1}t_y \\
\Rightarrow\ &\alpha^2[(s_x-t_x)^2+t_y^2] = (s_x-t_x)^2 + (\alpha^2-1)t_y^2 \\
&\qquad - 2(s_x-t_x)\sqrt{\alpha^2-1}t_y \\
&\Rightarrow 0 = \alpha^2(s_x-t_x)^2 - (s_x-t_x)^2 + \alpha^2 t_y^2 \\
&\qquad - \alpha^2 t_y^2 + t_y^2 + 2(s_x-t_x)\sqrt{\alpha^2-1}t_y \\
&\qquad = (\alpha^2-1)(s_x-t_x)^2 + t_y^2 \\
&\qquad + 2(s_x-t_x)^2\sqrt{\alpha^2-1}t_y \\
&\qquad = (t_y + \sqrt{\alpha^2-1}(s_x-t_x))^2 \\
&\Rightarrow t_y = -\sqrt{\alpha^2-1}(s_x-t_x).
\end{aligned}
$$

□

Lemma 23 *The bisector $b_{11,12}$ bisector is given by $y = -\frac{(s_x+x)+2\alpha\sqrt{s_x x}}{\sqrt{\alpha^2-1}}$.*

Proof. We want the curve defining the bisector between the region containing the points $t = (t_x,t_y)$ such that the shortest path from $s = (s_x,0)$ to t is $\pi_{11}(s,t)$, and the region containing the points $t = (t_x,t_y)$ such that the shortest path from $s = (s_x,0)$ to t is $\pi_{12}(s,t)$. Thus,

$$
\begin{aligned}
&\sqrt{\alpha^2-1}(s_x+t_x) - t_y = \alpha\sqrt{(s_x-t_x)^2+t_y^2} \\
\Rightarrow\ &\left[\sqrt{\alpha^2-1}(s_x+t_x) - t_y\right]^2 = \alpha^2((s_x-t_x)^2+t_y^2).
\end{aligned}
$$

By expanding the square we find

$$
\begin{aligned}
0 &= (\alpha^2-1)t_y^2 + 2\sqrt{\alpha^2-1}(s_x+t_x)t_y + \alpha^2(s_x-t_x)^2 \\
&\quad - (\alpha^2-1)(s_x+t_x)^2 \\
&= (\alpha^2-1)t_y^2 + 2\sqrt{\alpha^2-1}(s_x+t_x)t_y + \alpha^2 s_x^2 + \alpha^2 t_x^2 \\
&\quad - 2\alpha^2 s_x t_x - \alpha^2 s_x^2 - \alpha^2 t_x^2 - 2\alpha^2 s_x t_x + (s_x+t_x)^2 \\
&= (\alpha^2-1)t_y^2 + 2\sqrt{\alpha^2-1}(s_x+t_x)t_y - 4\alpha^2 s_x t_x \\
&\quad + (s_x+t_x)^2.
\end{aligned}
$$

From which we obtain that

$$
\begin{aligned}
t_y &= \frac{-2\sqrt{\alpha^2-1}(s_x+t_x)}{2(\alpha^2-1)} \\
&\quad \pm \frac{\sqrt{4(\alpha^2-1)((s_x+t_x)^2 - [-4\alpha^2 s_x t_x + (s_x+t_x)^2])}}{2(\alpha^2-1)} \\
&= \frac{-\sqrt{\alpha^2-1}(s_x+t_x) \pm \sqrt{(\alpha^2-1)4\alpha^2 s_x t_x}}{\alpha^2-1} \\
&= \frac{-(s_x+t_x) \pm 2\alpha\sqrt{s_x t_x}}{\sqrt{\alpha^2-1}}.
\end{aligned}
$$

Let $(0,b_1)$ and $(0,b_2)$ be, respectively, the points where $\pi_{12}(s,t)$ leaves and enters for the second time the region R. We know that $b_1 > b_2$. Thus, using Lemma 16, we know that $\pi_{12}(s,t)$ exists if $t_y < -\frac{s_x+t_x}{\sqrt{\alpha^2-1}}$. Hence, the bisector is given by the curve $t_y = \frac{-(s_x+t_x)-2\alpha\sqrt{s_x t_x}}{\sqrt{\alpha^2-1}}$. □

Computing shortest paths amid non-overlapping weighted disks*

Prosenjit Bose† Jean-Lou De Carufel‡ Guillermo Esteban§ Anil Maheshwari¶

Abstract

In this article, we present an approximation algorithm for solving the Weighted Region Problem amidst a set $\mathfrak{D}$ of n non-overlapping weighted disks in the plane. For a given parameter $\varepsilon \in (0, 1]$, the length of the approximate path is at most $(1+\varepsilon)$ times larger than the length of the actual shortest path. The algorithm is based on the discretization of the space by placing points on the boundary of the disks. Using such a discretization we can use Dijkstra's algorithm for computing a shortest path in the geometric graph obtained in (pseudo-)polynomial time.

1 Introduction

Computing a geodesic path (i.e., shortest path) between two points s and t in a geometric setting is one of the most studied problems in computational geometry. Applications of geometric shortest path problems are ubiquitous, appearing in diverse areas such as robotics [16, 25, 26], video games design [18, 29], or geographic information systems [13]. We refer to Mitchell [21] for an excellent survey on geometric shortest path problems.

In contrast to the classical shortest path problem in graphs, where the space of possible paths is discrete, in geometric settings the space is continuous: the source and target points can be anywhere within a certain geometric domain (e.g., a polygon, the plane, a surface), and the set of possible paths to consider has infinite size. Many variations of geometric shortest path problems have been studied before, depending on the geometric domain, the objective function (e.g., Euclidean metric, link-distance, geodesic distance), or specific domain constraints (e.g., obstacles in the plane, or holes in polygons). Finding shortest paths among polygonal obstacles in the plane has drawn great interest [3, 5, 22, 30]. Some of these results apply directly to real world problems. For instance, for modelling subdivisions of surfaces, embedding models use cylindrical faces, quadrics or patch together surfaces that are defined via bicubic or quadratic splines, see, e.g., [6, 11, 20]. Motion planning problems typically involve motion of curved objects through obstacles having curved boundaries [10, 19]. Modern font design systems rely upon conic and cubic spline curves [23, 24]. Numerous applications need efficient algorithms for processing curved objects directly [15, 28]. The way to tackle arbitrary real objects has been to approximate them as polygons or polyhedra of a sufficient number of vertices. This process is generally unsatisfactory, see [14]. For these reasons, in this paper, as in [7, 8, 9, 17], we focus on the problem of computing shortest paths among curved objects. In particular, we consider disks of different radii with a non-negative weight assigned to them.

1.1 Previous results

One of the most general versions of the shortest path problem that has been studied consists of a subdivision of the two-dimensional space. Without loss of generality, we assume it to be triangulated. Each region has a (non-negative) weight associated to it, representing the cost per unit distance of traveling in that region. Thus, the cost of traversing a region is typically given by the Euclidean distance traversed in the region, multiplied by the corresponding weight. The resulting metric is often called the *weighted region metric*, and the problem of computing a shortest path between two points under this metric is known as the *Weighted Region Problem* (WRP). This problem is very general, since it allows to model many well-known variants of geometric shortest path problems.

The WRP was first introduced by Mitchell and Papadimitriou [22]. They provided an approximation algorithm that computes a $(1 + \varepsilon)$-approximation path in $O\left(n^8 \log \frac{nNW}{w\varepsilon}\right)$ time, where n is the total number of vertices describing the polygonal regions, N is the maximum integer coordinate of any vertex of the subdivision, W (resp., w) is the maximum (resp., minimum non-zero) integer weight assigned to a face of the subdivision.

Recently, it has been shown that the WRP cannot be solved exactly within the Algebraic Computation Model over the Rational Numbers [12]. In this model one can

*Partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†School of Computer Science, Carleton University, jit@scs.carleton.ca

‡School of Electrical Engineering and Computer Science, University of Ottawa, jdecaruf@uottawa.ca

§Departamento de Física y Matemáticas, Universidad de Alcalá and School of Computer Science, Carleton University, g.esteban@uah.es

¶School of Computer Science, Carleton University, anil@scs.carleton.ca

Time complexity	Reference
$O\left(n^8 \log \frac{nNW}{w\varepsilon}\right)$	[22]
$O\left(N^4 \log\left(\frac{NW}{w\varepsilon}\right) \frac{n}{\varepsilon^2} \log \frac{nN}{\varepsilon}\right)$	[1]
$O\left(N^2 \log\left(\frac{NW}{w}\right) \frac{n}{\varepsilon} \log \frac{1}{\varepsilon} \left(\frac{1}{\sqrt{\varepsilon}} + \log n\right)\right)$	[2]
$O\left(N^2 \log\left(\frac{NW}{w}\right) \frac{n}{\varepsilon} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon}\right)$	[30]
$O\left(N^2 \log\left(\frac{NW}{w}\right) \frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon}\right)$	[3]

Table 1: Some $(1+\varepsilon)$-approximation algorithms for the WRP. In this table, N is the maximum integer coordinate of any vertex of the subdivision, W (resp., w) is the maximum (resp., minimum non-zero) integer weight assigned to a face of the subdivision.

compute exactly any number that can be obtained from rational numbers by a finite number of basic operations. This emphasizes the need for high-quality approximation paths instead of optimal paths.

Most of the results for the WRP are focused on polygonal obstacles. The most common scheme followed in the literature is to discretize the geometric space by positioning Steiner points, and then build a graph by connecting pairs of Steiner points, see [1, 2, 3, 5, 30]. An approximate solution is constructed by finding a shortest path in this graph, by using well-known combinatorial algorithms (e.g., Dijkstra's algorithm). See Table 1 for the time complexity of some approximation algorithms designed following this scheme.

However, we are aware of only a few publications that treat curved objects. In general, we do not seem to have a good grasp on the complexity of weighted shortest paths when the region boundaries are nonlinear curves. The particular case where we consider n pseudodisk obstacles in the two-dimensional space (i.e., the weight of all the regions is infinity) can be solved exactly in $O(n \log n + k)$ time, where k is the size of the extended visibility graph of the union of the pseudodisks [8]. Later, Chen and Wang [9] computed a shortest path avoiding a set S of h pairwise disjoint splinegons with a total of n vertices in $O(n + h \log h + d)$ time, where d is a parameter sensitive to the geometric structures of the input, by applying a bounded degree decomposition of the set of obstacles. This improves the result in [8] when $h = o(n)$.

Obstacles with curved boundaries present both algebraic and combinatorial challenges [7]. Thus, Hershberger et al. [17] proposed an $O(n \log n)$ time algorithm for the shortest path problem based on certain assumptions on the computation of locating the intersection of two bisectors defined by pairs of curved obstacle boundary segments. They provided a $(1+\varepsilon)$-approximation of a shortest path in $O\left(n \log n + n \log \frac{1}{\varepsilon}\right)$ time without the bisector computation assumption.

Moreover, if we consider the problem for weighted disks where inside each region we can travel between any pair of points at no cost whereas outside all regions the travel cost between two points is their Euclidean distance, this can be seen as a redefinition of the additively weighted point set spanner problem. Bose et al. [4] were able to show that it is possible to design a graph G with a linear number of edges such that for any pair of disks D and D' there is a path in G whose length is arbitrarily close to the Euclidean distance between D and D'. Recently, this was improved by Smid [27] by reducing the number of edges needed by a factor of 4.

1.2 Our results

Sometimes, the shape of a real-world curved object can be approximated using a polygon whose vertices are specified by a subset of c points on the object, where c is a sufficiently large value. Then, one approach to solving the WRP on a set of curved regions would be to approximate each region with a polygon, and then use existing algorithms that work on polygons. However, this method is not always optimal.

Let $\mathfrak{D} = \{D_1, \ldots, D_n\}$ be a set disks, each with a radius $R_i > 0$, and for any pair D_i and D_j, $1 \leq i < j \leq n$, $D_i \cap D_j = \emptyset$. In addition, each disk has a (non-negative) weight ω_i assigned to it. In this paper, we provide an algorithm to compute a path between two points amidst $\mathfrak{D}$ that is at most $(1+\varepsilon)$ times larger than the actual shortest path. To solve this problem, we use the traditional technique of partitioning the 2-dimensional space into a discrete space by using a non-trivial Steiner points placement and designing an appropriate graph. Without loss of generality, we may assume that s and t are vertices of this graph. In particular, the main results of this paper are:

- The special case where the weight of all the disks is at least $\frac{\pi}{2}$ can be solved exactly by using the algorithms in, e.g., [8, 9, 17]. See Section 2.
- For the general version of the WRP, we propose a discretization that consists of a set of Steiner points along the boundary of each disk. We first place some vertices, called *vertex vicinity centers*, evenly on the boundary of each disk. Then, if the weight of the disk is strictly positive, we create an annulus around each vertex vicinity center, and we place a set of Steiner points inside each annulus. For a given approximation parameter $\varepsilon \in (0, 1]$, the number of vertices of the discretization is at most $C(\mathfrak{D})\frac{n}{\varepsilon}$, where $C(\mathfrak{D})$ captures geometric parameters and the weights of $\mathfrak{D}$. See Section 3.
- We show that the weighted length of the approximated path between any pair of nodes is at most $(1+\varepsilon)$ times the weighted length of a shortest

path. This approximation path can be computed by executing shortest path algorithms on the graph formed by Steiner points where two Steiner points are joined by an edge. See Section 4.

2 Preliminaries

Any continuous (rectifiable) curve lying in the two-dimensional space is called a *path*. Let $\Pi(s,t)$ denote a path from a source point s to a target point t among $\mathfrak{D} = \{D_1, \ldots, D_n\}$. Let R_i and c_i be, respectively, the radius and the center of each disk D_i. Let $\omega_i \in \mathbb{R}_{\geq 0}, i \in \{1, \ldots, n\}$, be the weight associated to a disk $D_i \in \mathfrak{D}$, which represents the cost of traveling a unit Euclidean distance inside that disk. In addition, and without loss of generality, we can assume that the weight outside the disks is 1. Otherwise, we could always rescale the weights to be 1 outside the disks. Then, the weighted length of $\Pi(s,t)$ is given by $\|\Pi(s,t)\| = \mu + \sum_{i=1}^{n} \omega_i \cdot |\pi_i|$, where μ denotes the Euclidean length of the intersection between $\Pi(s,t)$ and the space outside the disks, and $|\pi_i|$ denotes the Euclidean length of the intersection between $\Pi(s,t)$ and a disk D_i, that is, $\pi_i = \Pi(s,t) \cap D_i$. In case π_i coincides with an arc of D_i, the weight of traveling along that arc is given by $\min\{1, \omega_i\}$. Given s and t, a weighted shortest path $SP_w(s,t)$ is a path that minimizes the weighted length between s and t.

Observe that every path consists of a sequence of (straight or circular-arc) segments whose endpoints $a_1, \ldots, a_m$ are on the boundary of the disks in $\mathfrak{D}$. These endpoints $a_1, \ldots, a_m$, are called *bending points*.

We now present some properties of a shortest path between two points on the boundary of the same disk that will be useful in the forthcoming sections. Observation 1 gives the (weighted) length of a subpath between two points p and q on the boundary of a disk $D \in \mathfrak{D}$. The result can be proved using the law of cosines.

Observation 1 *Let p and q be two consecutive bending points of the path $SP_w(s,t)$ on the boundary of $D \in \mathfrak{D}$ centered at c with radius R and weight $\omega \geq 0$. Let θ be the angle $\angle cpq$.*

- *If $SP_w(p,q)$ coincides with an arc of D, then $\|SP_w(p,q)\| = R \cdot (\pi - 2\theta)$; see the red path in Figure 1.*
- *If $SP_w(p,q)$ only intersects the boundary of D at p and q, then $\|SP_w(p,q)\| = \omega \cdot 2R\cos\theta$; see the blue path in Figure 1.*

The following result follows from the fact that the weight of the boundary of a region is given by the minimum among the weights of the two adjacent regions.

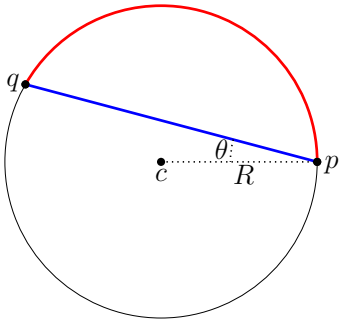


Figure 1: The two types of shortest paths between p and q on the boundary of a disk.

Observation 2 *Let p and q be two consecutive bending points of the path $SP_w(s,t)$ on the boundary of D. Let $\omega \in [0,1]$ be the weight of D. Then $SP_w(p,q)$ only intersects the boundary of D at p and q.*

Now consider a special case of the WRP where all the regions have a weight $\omega \geq \frac{\pi}{2}$. Lemma 1 states that when the weight of a disk is at least $\frac{\pi}{2}$, then the disk can be considered as an obstacle. Hence, if all disks have weight at least $\frac{\pi}{2}$, we can compute exactly a shortest path between any pair of points, see, e.g., [9].

Lemma 1 *Let p and q be two consecutive bending points of the path $SP_w(s,t)$ on the boundary of D. Let $\omega \geq \frac{\pi}{2}$ be the weight of D. Then $SP_w(p,q)$ coincides with the shortest arc of D from p to q.*

Proof. We need to prove that the weight of a path intersecting the interior of the disk is larger than when going along the boundary, i.e., that $R \cdot (\pi - 2\theta) \leq 2R\omega\cos\theta \iff \pi - 2\theta \leq 2\omega\cos\theta$, for any angle $\theta \in [0, \frac{\pi}{2}]$.

We know that $\omega \geq \frac{\pi}{2}$, so $2\omega\cos\theta \geq \pi\cos\theta$. Thus, it is sufficient to prove that $\pi - 2\theta \leq \pi\cos\theta$. We first minimize the function $\pi\cos\theta + 2\theta$:

$$\frac{\partial \pi\cos\theta + 2\theta}{\partial\theta} = -\pi\sin\theta + 2 = 0 \iff \sin\theta = \frac{2}{\pi}$$

For this value of θ, we get that $\pi\cos\theta + 2\theta \geq \sqrt{\pi^2 - 4} + 2\arcsin\frac{2}{\pi}$, which is greater than π, which gives us the desired result. □

Now, we state that there are no other ways for a shortest path between p and q to intersect the disk D than the ones described in Observation 1. This means that if p and q are two consecutive bending points on the boundary of D, a shortest path between p and q is a (straight or circular-arc) segment, i.e., $SP_w(p,q)$ does not bend on the boundary of D.

Lemma 2 *Let p and q be two consecutive bending points of the path $SP_w(s,t)$ on the boundary of D. If $\omega \in (1, \frac{\pi}{2})$, then a shortest path between p and q is a (straight or circular-arc) segment.*

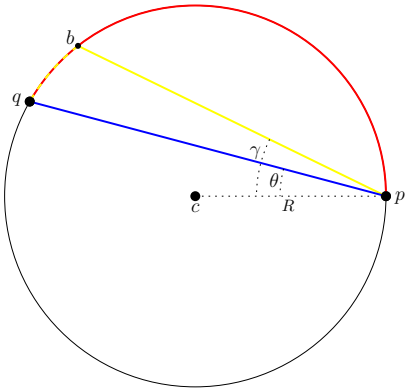


Figure 2: The two possible types of shortest paths between p and q on the boundary of a disk are depicted in red and blue.

Proof. Suppose there is a point $b \neq p, q$ on the boundary of D where $SP_w(p,q)$ bends, and let γ be the angle $\angle cpb$, see Figure 2.

In this case, the length of $SP_w(p,q)$ is $\|SP_w(p,q)\| = R\frac{\sin(\pi-2\gamma)}{\sin\gamma}\omega + 2R(\gamma-\theta) = R\cdot\left(\frac{2\cos\gamma\sin\gamma}{\sin\gamma}\omega + 2(\gamma-\theta)\right) = 2R\left(\omega\cos\gamma + (\gamma-\theta)\right)$.

The value $\|SP_w(p,q)\|$ is minimized when $\cos\gamma = \frac{\sqrt{\omega^2-1}}{\omega}$:

$$\frac{\partial\|SP_w(p,q)\|}{\partial\gamma} = -2R\omega\sin\gamma + 2R = 0 \iff \omega\sin\gamma = 1$$
$$\iff \begin{cases}\sin\gamma = \frac{1}{\omega},\\ \cos\gamma = \frac{\sqrt{\omega^2-1}}{\omega}.\end{cases}$$

We can see that the equation holds since $\omega \in (1, \frac{\pi}{2})$. Hence, for this value of γ, the weighted length of a shortest path from p to q is:

$$\|SP_w(p,q)\| = 2R\left(\omega\cos\gamma + (\gamma-\theta)\right)$$
$$= 2R\sqrt{\omega^2-1} + 2R\left(\arcsin\left(\frac{1}{\omega}\right) - \theta\right).$$

Now, we need to compare the length of $SP_w(p,q)$ with the length of (i) a path $\pi_1(p,q)$ that only intersects the boundary of D at p and q, and (ii) a path $\pi_2(p,q)$ along the boundary of D.

We first define the function $\omega\cos\theta + \theta$ that allows us to prove that $\|\pi_1(p,q)\| \leq \|SP_w(p,q)\|$. The maximum value of the function, when $\theta \in \left(0, \frac{\pi}{2}\right]$ is obtained next:

$$\frac{\partial\omega\cos\theta+\theta}{\partial\theta} = -\omega\sin\theta + 1 = 0 \iff \omega\sin\theta = 1$$
$$\iff \sin\theta = \frac{1}{\omega} \iff \theta = \arcsin\left(\frac{1}{\omega}\right).$$

Hence,

$$\omega\cos\theta + \theta \leq \omega\cos\left(\arcsin\left(\frac{1}{\omega}\right)\right) + \arcsin\left(\frac{1}{\omega}\right)$$
$$= \sqrt{\omega^2-1} + \arcsin\left(\frac{1}{\omega}\right)$$
$$\implies \omega\cos\theta \leq \sqrt{\omega^2-1} + \arcsin\left(\frac{1}{\omega}\right) - \theta$$
$$\implies 2R\omega\cos\theta \leq 2R\sqrt{\omega^2-1} + 2R\left(\arcsin\left(\frac{1}{\omega}\right) - \theta\right)$$
$$\implies \|\pi_1(p,q)\| \leq \|SP_w(p,q)\|.$$

Now, we define another function $\omega^5 - 14\omega^3 + 37\omega$ that allows us to prove that $\|\pi_2(p,q)\| \leq \|SP_w(p,q)\|$. The maximum of this function, when $\omega \in (1, \frac{\pi}{2})$, is obtained next:

$$\frac{\partial\omega^5 - 14\omega^3 + 37\omega}{\partial\omega} = 5\omega^4 - 42\omega^2 + 37 = 0 \iff \omega = 1.$$

Hence,

$$\omega^5 - 14\omega^3 + 37\omega \leq 24 \implies \frac{\omega^4 - 14\omega^2 + 37}{24} \leq \frac{1}{\omega}$$
$$\implies \frac{24 - 12\omega^2 + 12 + \omega^4 - 2\omega^2 + 1}{24} \leq \frac{1}{\omega}$$
$$\implies 1 - \frac{\omega^2-1}{2} + \frac{(\omega^2-1)^2}{24} \leq \frac{1}{\omega}.$$

The Taylor series of the function $\cos x$ is $\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}x^{2n}$, for all x. Thus, $\cos\sqrt{\omega^2-1} \leq 1 - \frac{\omega^2-1}{2} + \frac{(\omega^2-1)^2}{24}$, and we get that

$$\cos\sqrt{\omega^2-1} \leq \frac{1}{\omega} \implies \sin\left(\frac{\pi}{2} - \sqrt{\omega^2-1}\right) \leq \frac{1}{\omega}$$
$$\implies \frac{\pi}{2} - \sqrt{\omega^2-1} \leq \arcsin\left(\frac{1}{\omega}\right)$$
$$\implies \pi - 2\sqrt{\omega^2-1} \leq 2\arcsin\left(\frac{1}{\omega}\right)$$
$$\implies R\cdot(\pi - 2\theta) \leq 2R\left(\sqrt{\omega^2-1} + \arcsin\left(\frac{1}{\omega}\right) - \theta\right)$$
$$\implies \|\pi_2(p,q)\| \leq \|SP_w(p,q)\|.$$

We proved that the length of both paths $\pi_1(p,q)$ and $\pi_2(p,q)$ is not larger than the length of the path $SP_w(p,q)$. Hence, a shortest path from p to q is either the straight-line segment from p to q or a shortest arc of D from p to q. □

3 Discretization

In this section, we construct a weighted graph $G_\varepsilon = (V_\varepsilon(G_\varepsilon), E_\varepsilon(G_\varepsilon))$ by carefully adding Steiner points on the boundary of the disks. Then, one can apply Dijkstra's algorithm on G_ε to obtain a path $\tilde{\pi}(s,t)$ that is a $(1+\varepsilon)$-approximation of $SP_w(s,t)$.

Lemma 1 states that when the weight of a disk is at least $\frac{\pi}{2}$, then no shortest path will intersect the interior of that disk. In this paper we are discretizing the

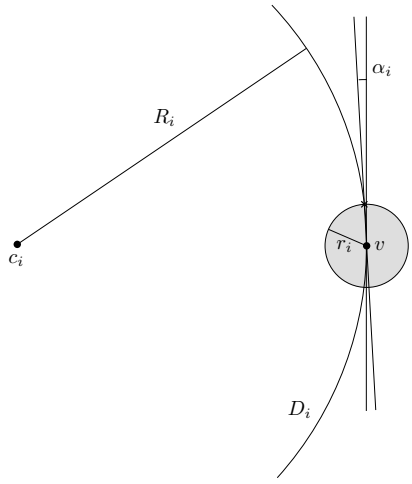


Figure 3: Vertex vicinity of a vertex v (in grey) on the boundary of disk D_i.

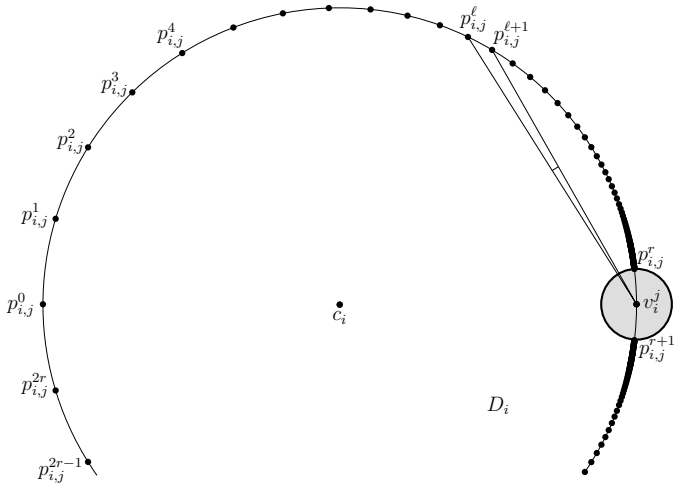


Figure 4: Ring points associated to the vertex vicinity center v_i^j on the boundary of D_i.

space to obtain a $(1+\varepsilon)$-approximation of a shortest path. Thus, from now on, we can assume, without loss of generality, that the maximum weight of the regions is $\frac{\pi}{2}$.

First, we introduce the value d_i defined as the minimum Euclidean distance from D_i to any other disk D_j in $\mathfrak{D}$. We also define a *weighted angular radius* α_i to be $\alpha_i = \arcsin\left(\frac{\min\{d_i,R_i\}\min\{1,\omega_i\}}{4R_i\max\{1,\omega_i\}}\right)$. Observe that α_i is not larger than $\frac{\pi}{2}$.

Definition 1 *Let v be a point on the boundary of $D_i \in \mathfrak{D}$. We refer to the disk with center v and radius $2R_i\sin\alpha_i$ as the* vertex vicinity of vertex v*, or* vertex vicinity *when v is clear from the context.*

Observe that in Definition 1, α_i is equal to the minimum angle between (i) the line tangent to D_i at v, and (ii) the line through v and the intersection point between the vertex vicinity of v and D_i. See Figure 3. We use the definition of vertex vicinity to place k_i points around each disk D_i. These points $\{v_i^1,\ldots,v_i^{k_i}\}$, called *vertex vicinity centers*, are equally spaced around D_i.

If the weight of the disk D_i is 0, we define the angular distance between c_i, and two consecutive vertex vicinity centers v_i^ℓ and $v_i^{\ell+1}$, for $1 \le \ell < k_i$, by

$$\angle v_i^\ell c_i v_i^{\ell+1} = \frac{\varepsilon d_i}{a(d_i+1)}, \text{ for } \varepsilon \in \left(0, \frac{a\pi}{2\omega_i}\right], \qquad (1)$$

where $a = \frac{1+3c+\sqrt{9c^2+10c+1}}{2}$, $c = \frac{\frac{\pi}{2}\max_{1\le j\le n}\{R_j\}}{\min_{1\le j\le n}\{d_j\}}$, and k_i is the largest integer satisfying $\angle v_i^1 c_i v_i^{k_i} < 2\pi$. Note that we do not consider the particular case where the disks overlap, so $c > 0$, and $a > 1$.

Otherwise, if the weight of D_i is $\omega_i > 0$, we have that $k_i = \left\lfloor \frac{\pi}{2\alpha_i} \right\rfloor$. Let $p_{i,j}^0$ be the point diametrically opposed to v_i^j in D_i. We associate to each vertex vicinity center v_i^j a set of $2r+1$ points on D_i, noted as $p_{i,j}^0,\ldots,p_{i,j}^{2r}$, each of which is called a *ring point*. The ring points $p_{i,j}^0,\ldots,p_{i,j}^r$ are placed on the boundary of the half disk to the right of $\overrightarrow{v_i^j p_{i,j}^0}$, from $p_{i,j}^0$ to v_i^j in clockwise order. The points $p_{i,j}^{r+1},\ldots,p_{i,j}^{2r}$ are placed symmetrically on the other half of the disk. We impose the condition that inside the vertex vicinity of each vertex v_i^j, we do not place ring points associated to v_i^j. Thus, r is the largest integer satisfying $\angle p_{i,j}^0 v_i^j p_{i,j}^r \le \frac{\pi}{2} - \alpha_i$. See Figure 4 for an illustration of the ring points associated to v_i^j. We define the angular distance between v_i^j, and two consecutive ring points $p_{i,j}^\ell$ and $p_{i,j}^{\ell+1}$, for $0 \le \ell < r$, by

$$\angle p_{i,j}^\ell v_i^j p_{i,j}^{\ell+1} = \frac{\omega_i\varepsilon}{a}\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^\ell, \text{ for } \varepsilon \in \left(0, \frac{a\pi}{2\omega_i}\right]. \qquad (2)$$

Based on previous discretization schemes (e.g., [2, 3]), and without loss of generality, we may assume that the points s and t are vertex vicinity centers or ring points.

From Equation (2), and using the definition of vertex vicinity and the formula for the sum of the first terms of a geometric series, we can obtain the number of ring points associated to v_i^j that we are adding to the boundary of a disk D_i with weight $\omega_i > 0$.

Proposition 3 *Let D_i be a disk with weight $\omega_i > 0$. The number of ring points associated to v_i^j inserted on D_i is upper-bounded by $2\frac{1+\log_2\frac{\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)}+1$.*

Proof. By the definition of the ring points we know that the angular distance between consecutive Steiner points on half a disk is $\angle p_{i,j}^\ell v_i^j p_{i,j}^{\ell+1} = \frac{\omega_i\varepsilon}{a}\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^\ell$. So the angular distance from point $p_{i,j}^0$ to each of the

Steiner points $p^{\ell}_{i,j},\ \ell \in \{1,\ldots,r\}$ is given by

$$\begin{aligned}\angle p^0_{i,j} v^j_i p^{\ell}_{i,j} &= \sum_{m=0}^{\ell-1} \frac{\omega_i \varepsilon}{a}\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^m \\ &= \frac{\omega_i\varepsilon}{a}\cdot\frac{1-\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^{\ell}}{1-\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)} \\ &= \frac{\omega_i\varepsilon}{a}\cdot\frac{1-\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^{\ell}}{\frac{2\omega_i\varepsilon}{a\pi}} \\ &= \frac{\pi}{2}\left(1-\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^{\ell}\right).\end{aligned}$$

The largest value of r such that $\angle p^0_{i,j} v^j_i p^r_{i,j} \leq \frac{\pi}{2}-\alpha_i$ can be found by solving the following inequality:

$$\begin{aligned}&\frac{\pi}{2}\left(1-\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^{r}\right) \leq \frac{\pi}{2}-\alpha_i \\ \implies &1-\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^{r} \leq 1-\frac{2\alpha_i}{\pi} \\ \implies &\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)^{r} \geq \frac{2\alpha_i}{\pi} \\ \implies &r \leq \log_{1-\frac{2\omega_i\varepsilon}{a\pi}} \frac{2\alpha_i}{\pi} \\ = &\frac{\log_2 \frac{2\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)} = \frac{1+\log_2\frac{\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)}.\end{aligned}$$

Since we need to add ring points around the whole disk D_i, we need two times the number of points in the previous equation. In addition, we need to take into account vertex $p^0_{i,j}$, hence the final result. □

Observe that we do not need to place the number of points from Proposition 3 around each vertex vicinity center v^j_i, since some of them are further away than the ring points associated to neighboring vertex vicinity centers. Hence, we create an annulus around each point v^j_i and we place ring points only inside these annuli. The smallest circumference of the annuli, i.e., the boundary of the vertex vicinity of v^j_i, has radius $2R_i \sin\alpha_i$, and the largest circumference has radius $2R_i \sin(2\alpha_i)$. An upper bound on the total number of points placed on each disk D_i with weight $\omega_i > 0$ is given next.

Proposition 4 *Let D_i be a disk with weight $\omega_i > 0$. The total number of vertex vicinity centers and ring points added on D_i is upper-bounded by $\frac{1}{\log_2 \frac{a\pi}{a\pi-2\omega_i\varepsilon}}\frac{\pi}{\alpha_i}$.*

Proof. Let v^j_i be a vertex vicinity center on the boundary of a disk D_i, for $j \in \{1,\ldots,\frac{\pi}{2\alpha_i}\}$. By Proposition 3 we place $2\frac{1+\log_2\frac{\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)}+1$ points associated to v^j_i on the boundary of D_i. Since we are creating an annulus where the largest circumference has radius $2\cos\alpha_i$ times the radius of the small one (see Figure 5), outside this second disk, and around D_i, we are placing $2\frac{1+\log_2\frac{2\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)}+1$ points.

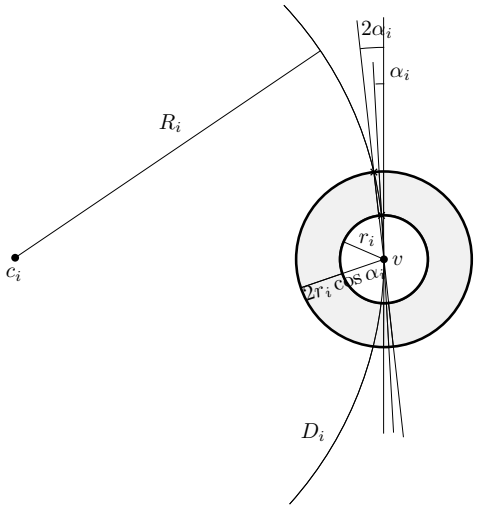


Figure 5: The annulus where the ring points from v are placed is represented in grey.

Then, the idea is to calculate the number of Steiner points inside the annulus:

$$\begin{aligned}&2\left(\frac{1+\log_2\frac{\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)}-\frac{1+\log_2\frac{2\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)}\right) \\ &= 2\frac{\log_2\frac{\alpha_i}{\pi}-\log_2\frac{2\alpha_i}{\pi}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)} = \frac{2\log_2\frac{\frac{\alpha_i}{\pi}}{\frac{2\alpha_i}{\pi}}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)} \\ &= \frac{2\log_2\frac{1}{2}}{\log_2\left(1-\frac{2\omega_i\varepsilon}{a\pi}\right)} = \frac{-2}{\log_2\frac{a\pi-2\omega_i\varepsilon}{a\pi}} = \frac{2}{\log_2\frac{a\pi}{a\pi-2\omega_i\varepsilon}}.\end{aligned}$$

Note that in the previous equation we are not counting the intersection points between D_i and the largest circumference around v^j_i. In addition, the intersection points between D_i and the smallest circumference around v^j_i coincide with the vertex vicinity centers of v^{j-1}_i and v^{j+1}_i. Finally, since the vertex vicinity centers belong to the set $V_\varepsilon(G_\varepsilon)$ of nodes, and we have $\frac{\pi}{2\alpha_i}$ vertex vicinity centers, we get the desired result. □

The total number of nodes in G_ε is $O(\frac{n}{\varepsilon})$, and the total number of edges is $O(\frac{n^2}{\varepsilon^2})$, where the constants hidden in the "big-O" notation depend on the geometric parameters and weights of $\mathfrak{D}$, see the appendix. In case that two nodes are not visible or are adjacent on the boundary of a disk, we join them by arcs of the disks, instead of using a straight-line segment. This way, we ensure that if a shortest path between a pair of points does not intersect the interior of the disks, then our algorithm computes a shortest path exactly.

4 Discrete path

If the source point s and the target point t are on the boundary of the same disk D and the only disk $SP_w(s,t)$ intersects is D, then we can compute $SP_w(s,t)$ exactly, and in constant time. This result is obtained by taking into account that there are only two possible shortest paths from s to t, see Observation 1.

Now, suppose s is on the boundary of a disk D centered at c, and t is outside D. We prove that there is a path $\tilde{\pi}(s,t)$ whose length is at most $\left(1+\frac{\varepsilon}{a}\right)$ times larger than the length of a shortest path from s to t when intersecting only D. This path $\tilde{\pi}(s,t)$ is a shortest path through the vertices of the discretization. In fact, we can compute a shortest path exactly in this case. However, the result in Lemma 5 will be useful to prove the approximation ratio when a shortest path intersects more than one disk.

Lemma 5 *Let s be a point on the boundary of a disk D centered at c and weight $\omega \geq 0$, and let t be a point outside D. If D is the only disk intersected by $SP_w(s,t)$, then $\|\tilde{\pi}(s,t)\| \leq \left(1+\frac{\varepsilon}{a}\right) \cdot \|SP_w(s,t)\|$.*

Proof. First, we will prove the case where $\omega > 0$. Suppose that a shortest path from s to t does not intersect the interior of D. In this case, the approximate shortest path intersects the arc of the disk from s to the tangency point from t to D. Before this tangency point, and along the boundary of D, there is a ring point. This ring point is joined to t by an edge which is not a straight-line segment. This means that in this case, an approximate shortest path is also a shortest path. Hence, $\|\tilde{\pi}(s,t)\| = \|SP_w(s,t)\| \leq \left(1+\frac{\varepsilon}{a}\right)\|SP_w(s,t)\|$.

Now, suppose that $SP_w(s,t)$ intersects the interior of D. Let q be the point where $SP_w(s,t)$ leaves the disk and let θ be the angle $\angle csq$. Let p be the closest ring point to q on the boundary of the disk, and let $\angle csp$ be $\theta+\varepsilon'$, for some $\varepsilon' \geq 0$ and $\theta < \frac{\pi}{2}$. The case where $\varepsilon' < 0$ will be addressed later in the proof. In addition, we can assume, without loss of generality, that D has radius length 1. Let s' be the point diametrically opposed to s on the boundary of D, and let γ be the angle $\angle s'ct$, see Figure 6.

Then, the length of the approximate path is

$$\begin{aligned}
&\|\tilde{\pi}(s,t)\| = 2\omega\cos(\theta+\varepsilon') \\
&+ \sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)} \\
&= \left(2\omega\cos\theta + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\right) \\
&\cdot \left[\frac{2\omega\cos(\theta+\varepsilon') + \sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)}}{2\omega\cos\theta + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}}\right].
\end{aligned}$$

We would like to prove that:

$$\frac{2\omega\cos(\theta+\varepsilon') + \sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)}}{2\omega\cos\theta + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}}$$

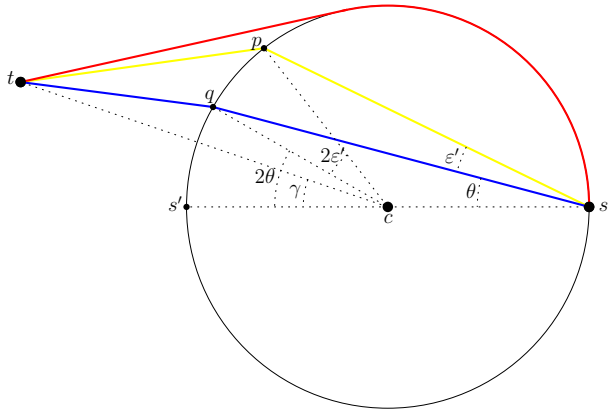


Figure 6: Notation when t is outside the disk.

$$\leq \frac{\omega\cos\theta+\varepsilon'}{\omega\cos\theta}.$$

Since $\cos(\theta+\varepsilon') \leq \cos\theta$, it is sufficient to prove that:

$$\begin{aligned}
&\frac{2\omega\cos\theta + \sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)}}{2\omega\cos\theta + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}} \\
&\leq \frac{\omega\cos\theta+\varepsilon'}{\omega\cos\theta}.
\end{aligned}$$

Thus,

$$\begin{aligned}
&2\omega^2\cos^2\theta + \omega\cos\theta\sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)} \\
&\leq \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\cdot(\omega\cos\theta+\varepsilon') \\
&\quad + 2\omega^2\cos^2\theta + 2\varepsilon'\omega\cos\theta \\
&\iff \omega\cos\theta\sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)} \\
&\leq \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}(\omega\cos\theta+\varepsilon') \\
&+ 2\varepsilon'\omega\cos\theta \\
&\iff \Big(\sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)} \\
&\quad - 2\varepsilon' - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\Big)\omega\cos\theta \\
&\leq \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\varepsilon'.
\end{aligned}$$

The term to the right of the inequality represents the length of a shortest path from s to t outside the disk, multiplied by a positive value ε', so this value is positive. In addition, $\omega > 0$, and since $\theta < \frac{\pi}{2}$, $\cos\theta > 0$. Then, it is enough to prove that

$$\begin{aligned}
&\sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)} - 2\varepsilon' \\
&-\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)} \leq 0 \\
&\iff \sqrt{1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma)} \\
&\quad \leq 2\varepsilon' + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)} \\
&\iff 1+|ct|^2-2|ct|\cos(2(\theta+\varepsilon')-\gamma) \leq 4\varepsilon'^2+1 \\
&\quad + |ct|^2-2|ct|\cos(2\theta-\gamma)
\end{aligned}$$

$$
\begin{aligned}
&+ 4\varepsilon'\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\\
\iff &-2|ct|\cos(2(\theta+\varepsilon')-\gamma) \le 4\varepsilon'^2 - 2|ct|\cos(2\theta-\gamma)\\
&+ 4\varepsilon'\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\\
\iff &2|ct|(\cos(2\theta-\gamma)-\cos(2(\theta+\varepsilon')-\gamma))\\
&\le 4\varepsilon'^2 + 4\varepsilon'\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}.
\end{aligned}
$$

Since $\cos a - \cos b = -2\sin\frac{a+b}{2}\sin\frac{a-b}{2}$ for all angles a, b, the previous inequality is equivalent to:

$$
\begin{aligned}
&-|ct|\sin\left(\frac{4\theta-2\gamma+2\varepsilon'}{2}\right)\sin(-\varepsilon')\\
&\le \varepsilon'^2 + \varepsilon'\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}.
\end{aligned}
$$

We know that $\sin a \le a$, so it is sufficient to prove that:

$$
\begin{aligned}
|ct|\sin(2\theta-\gamma+\varepsilon')\varepsilon' &\le \varepsilon'^2 + \varepsilon'\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\\
|ct|\sin(2\theta-\gamma+\varepsilon') &\le \varepsilon' + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\\
|ct|\sin(2\theta-\gamma+\varepsilon') - \varepsilon' &\le \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}.
\end{aligned}
$$

Now, ε' is just on the left-hand side of the inequality, so we would like to know what is the largest value $|ct|\sin(2\theta-\gamma+\varepsilon') - \varepsilon'$ can take

$$
\begin{aligned}
&\frac{\partial |ct|\sin(2\theta-\gamma+\varepsilon') - \varepsilon'}{\partial \varepsilon'} = |ct|\cos(2\theta-\gamma+\varepsilon') - 1 = 0\\
&\iff |ct|\cos(2\theta-\gamma+\varepsilon') = 1\\
&\iff \varepsilon' = \arccos\left(\frac{1}{|ct|}\right) + \gamma - 2\theta.
\end{aligned}
$$

We know that $\varepsilon' \ge 0$, so the maximum value is obtained when $\varepsilon' = \max\left\{0, \arccos\left(\frac{1}{|ct|}\right) + \gamma - 2\theta\right\}$. Hence,

- If $\varepsilon' = 0$, it is sufficient to prove that $|ct|\sin(2\theta-\gamma) \le \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}$:

$$
\begin{aligned}
&|ct|^2\sin^2(2\theta-\gamma) \le 1+|ct|^2-2|ct|\cos(2\theta-\gamma)\\
\iff &|ct|^2\sin^2(2\theta-\gamma) - |ct|^2 \le 1-2|ct|\cos(2\theta-\gamma)\\
\iff &-|ct|^2\cos^2(2\theta-\gamma) \le 1-2|ct|\cos(2\theta-\gamma)\\
\iff &1-2|ct|\cos(2\theta-\gamma)+|ct|^2\cos^2(2\theta-\gamma) \ge 0\\
\iff &(1-|ct|\cos(2\theta-\gamma))^2 \ge 0.
\end{aligned}
$$

- If $\varepsilon' = \arccos\left(\frac{1}{|ct|}\right) + \gamma - 2\theta$, it is sufficient to prove that $|ct|\sin\left(\arccos\left(\frac{1}{|ct|}\right)\right) - \arccos\left(\frac{1}{|ct|}\right) - \gamma + 2\theta \le \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}$:

$$
\begin{aligned}
&|ct|\sqrt{1-\frac{1}{|ct|^2}} - \arccos\left(\frac{1}{|ct|}\right) - \gamma + 2\theta\\
&\le \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\\
\iff &\sqrt{|ct|^2-1} - \arccos\left(\frac{1}{|ct|}\right) - \gamma + 2\theta\\
&\le \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\\
\iff &2\theta - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\\
&\le \arccos\left(\frac{1}{|ct|}\right) + \gamma - \sqrt{|ct|^2-1}. \qquad (3)
\end{aligned}
$$

Now, θ is just on the left-hand side of the inequality, so we would like to know which is the largest value $2\theta - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}$ can take:

$$
\begin{aligned}
&\frac{\partial\ 2\theta - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}}{\partial\theta}\\
&= 2 - \frac{2|ct|\sin(2\theta-\gamma)}{\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}} = 0\\
&\iff \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)} = |ct|\sin(2\theta-\gamma)\\
&\iff 1+|ct|^2-2|ct|\cos(2\theta-\gamma) = |ct|^2\sin^2(2\theta-\gamma)\\
&\iff 1 = |ct|\cos(2\theta-\gamma).
\end{aligned}
$$

Hence, $2\theta - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)} \le \arccos\left(\frac{1}{|ct|}\right) + \gamma - \sqrt{1+|ct|^2-2} = \arccos\left(\frac{1}{|ct|}\right) + \gamma - \sqrt{|ct|^2-1}$, which is (3), and that is what we wanted to prove.

In both cases, we proved that $\|\tilde{\pi}(s,t)\| \le \frac{\omega\cos\theta+\varepsilon'}{\omega\cos\theta}\|SP_w(s,t)\| = (1+\frac{\varepsilon'}{\omega\cos\theta})\|SP_w(s,t)\|$. We also know that $\cos\theta \ge 1-\frac{2}{\pi}\theta$ when $\theta \le \frac{\pi}{2}$. Hence,

$$
1 + \frac{\varepsilon'}{\omega\cdot\cos\theta} \le 1 + \frac{\varepsilon'}{\omega\left(1-\frac{2}{\pi}\theta\right)} = 1 + \frac{\pi\varepsilon'}{\omega(\pi-2\theta)}.
$$

However, we are interested in obtaining a $\left(1+\frac{\varepsilon}{a}\right)$-approximation. We can obtain this approximation factor by setting ε' to $\frac{\varepsilon\omega(\pi-2\theta)}{a\pi}$. Note that ε' represents the maximum angular distance between consecutive ring points. So, if we place the first Steiner point on the boundary of the disk, diametrically opposed to s, $\theta = 0$ and, in order to get a $\left(1+\frac{\varepsilon}{a}\right)$-approximation, we would like to have the following Steiner point at a distance $\frac{\varepsilon\omega}{a}$ from the first Steiner point, which is true by Equation (2). Following this procedure, one can prove that we always obtain a $\left(1+\frac{\varepsilon}{a}\right)$-approximation.

We also need to calculate the ratio when the closest ring point to q is to the left of q with respect to $SP_w(s,t)$. Suppose p' is the closest ring point to q on the boundary of the disk where $\angle csp' = \theta + \varepsilon'$ and $\varepsilon' \le 0$. If the approximation path through p is shorter than through p', the algorithm that calculates the approximation path will never go through p', so we do not need to calculate the ratio of the approximation path through p'. Otherwise, since the length of the approximation path is on the numerator, and the length of the shortest path does not change, the ratio when taking the approximate path

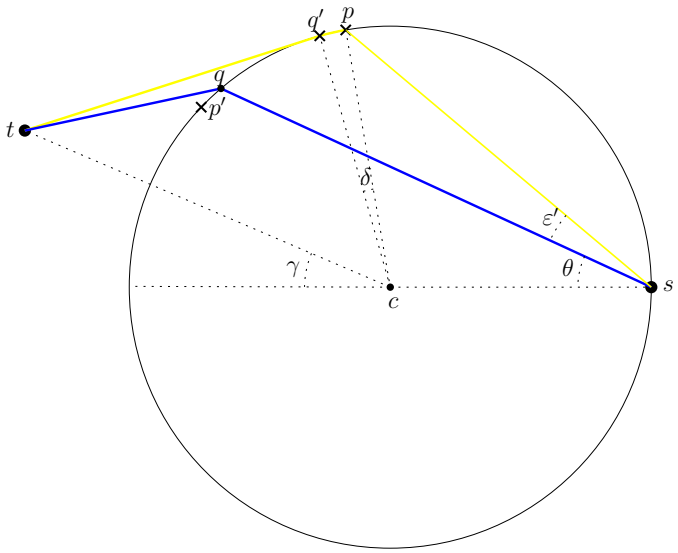


Figure 7: The tangency point from t to the disk D is after the Steiner point.

through p is larger. Also, note that in this case, if p is before the tangency point q' from t to D, the segment from p to t will intersect the interior of the disk, see Figure 7. Hence, we need to calculate the ratio in this particular case.

The length of the approximate path is given by $\|\tilde{\pi}(s,t)\| = \omega|sp| + \delta + |q't|$, where $\delta = \angle pcq'$, see the yellow path in Figure 7. We know from before that $\omega|sp| = 2\omega\cos(\theta+\varepsilon')$. We also know that $|q't| = \sqrt{|ct|^2-1}$, since q' is the tangency point from t to D. In addition, by using some trigonometric identities, we obtain that $\delta = 2(\theta+\varepsilon') - \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) - \gamma$. Thus,

$$\|\tilde{\pi}(s,t)\| = 2\omega\cos(\theta+\varepsilon') + \sqrt{|ct|^2-1} + 2(\theta+\varepsilon') - \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) - \gamma.$$

Our goal is to prove that:

$$\frac{\|\tilde{\pi}(s,t)\|}{2\omega\cos\theta + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}} \le \frac{\omega\cos\theta+\varepsilon'}{\omega\cos\theta}.$$

Since $\cos(\theta+\varepsilon') \le \cos\theta$, it is sufficient to prove that:

$$\frac{2(\omega\cos\theta+\theta+\varepsilon') - \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) - \gamma + \sqrt{|ct|^2-1}}{2\omega\cos\theta + \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}} \le \frac{\omega\cos\theta+\varepsilon'}{\omega\cos\theta}$$

$$\iff \left(2\omega\cos\theta + 2(\theta+\varepsilon') - \gamma - \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) + \sqrt{|ct|^2-1} - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\right)\omega\cos\theta \le \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}\varepsilon'.$$

We know that the term on the right-hand side of the inequality is positive, also $\omega > 0$, and since $\theta < \frac{\pi}{2}$, $\cos\theta > 0$. Then, it is sufficient to prove that

$$A = 2\theta - \gamma - \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) + \sqrt{|ct|^2-1} - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)} \le 0.$$

We now maximize the function A with respect to θ:

$$\frac{\partial A}{\partial\theta} = 2 - \frac{4|ct|\sin(2\theta-\gamma)}{2\sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}} = 0$$
$$\iff |ct|\sin(2\theta-\gamma) = \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)}$$
$$\iff |ct|^2\sin^2(2\theta-\gamma) = |ct|^2+1-2|ct|\cos(2\theta-\gamma)$$
$$\implies -|ct|^2\cos^2(2\theta-\gamma) = 1-2|ct|\cos(2\theta-\gamma)$$
$$\iff (1-|ct|\cos(2\theta-\gamma))^2 = 0$$
$$\implies \theta = \frac{\arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right)+\gamma}{2}.$$

Thus,

$$2\theta - \gamma - \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) + \sqrt{|ct|^2-1} - \sqrt{1+|ct|^2-2|ct|\cos(2\theta-\gamma)} \le \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) - \arcsin\left(\frac{\sqrt{|ct|^2-1}}{|ct|}\right) + \sqrt{|ct|^2-1} - \sqrt{1+|ct|^2-2|ct|\frac{1}{|ct|}} = 0.$$

This proves that $\|\tilde{\pi}(s,t)\| \le \left(1+\frac{\varepsilon}{a}\right)\|SP_w(s,t)\|$ when s is on the boundary of a disk D with positive weight.

In the special case where $\omega = 0$, we know the exact weight of a shortest path since $\pi - 2\theta + \gamma = \pi \implies 2\theta = \gamma$. Hence, $\|\tilde{\pi}(s,t)\| = \sqrt{1+|ct|^2-2|ct|\cos 2\varepsilon'}$, and $\|SP_w(s,t)\| = |ct|-1$. Thus,

$$\frac{\|\tilde{\pi}(s,t)\|}{\|SP_w(s,t)\|} = \frac{\sqrt{1+|ct|^2-2|ct|\cos 2\varepsilon'}}{|ct|-1} \le \frac{\sqrt{1+|ct|^2-2|ct|+4|ct|\varepsilon'^2}}{|ct|-1} = \sqrt{\frac{(|ct|-1)^2+4|ct|\varepsilon'^2}{(|ct|-1)^2}} = \sqrt{1+\frac{4|ct|\varepsilon'^2}{(|ct|-1)^2}}.$$

Now, we would like to prove that $\sqrt{1+\frac{4|ct|\varepsilon'^2}{(|ct|-1)^2}} \le 1+\frac{\varepsilon}{a} \implies 1+\frac{4|ct|\varepsilon'^2}{(|ct|-1)^2} \le \left(1+\frac{\varepsilon}{a}\right)^2$. We know that $|ct| \ge$

$d+1$, and $\varepsilon' \leq \frac{\varepsilon d}{2a(d+1)}$, where d is the minimum distance from D to any other disk. Thus, since $\frac{4|ct|\varepsilon'^2}{(|ct|-1)^2}$ is a decreasing function for $|ct| > 1$, we have that

$$\begin{aligned}1 + \frac{4|ct|\varepsilon'^2}{(|ct|-1)^2} &\leq 1 + \frac{4(d+1)\varepsilon'^2}{d^2} \\ &\leq 1 + \frac{4(d+1)\left(\frac{\varepsilon d}{2a(d+1)}\right)^2}{d^2} \\ &= 1 + \frac{\varepsilon^2}{a^2(d+1)} \leq 1 + \frac{\varepsilon^2}{a^2} \leq \\ &1 + \frac{\varepsilon^2}{a^2} + \frac{2\varepsilon}{a} = \left(1 + \frac{\varepsilon}{a}\right)^2. \qquad \square\end{aligned}$$

Next, we generalize Lemma 5 to the case where $SP_w(s,t)$ intersects an ordered sequence of disks of $\mathfrak{D}$. Recall that each disk D_i is centered at c_i, has radius R_i, and the weight inside the disk is $\omega_i \geq 0$, for $1 \leq i \leq n$. In addition, G_ε is the graph whose vertex set is the set of vertex vicinity centers and ring points, and each pair of points is joined by an edge, see Section 3.

Theorem 6 *Let $SP_w(s,t)$ be a weighted shortest path between two points s and t. There exists a path $\tilde{\pi}(s,t)$ in G_ε such that $\|\tilde{\pi}(s,t)\| \leq (1+\varepsilon) \cdot \|SP_w(s,t)\|$.*

Proof. Let $\mathcal{D} = (D_j, \ldots, D_k)$ be the ordered sequence of disks intersected by $SP_w(s,t)$. We can suppose that $s \in D_j$, and $t \in D_k$. The ordered sequence of points where $SP_w(s,t)$ enters the disks is given by $(s = a_1, a_2, \ldots, a_{k-j+1} = t)$, see Figure 8. The portions $SP_w(a_i, a_{i+1})$ are called *inter-vertex vicinity portions*.

By Observation 1, the subpaths $SP_w(a_i, a_{i+1})$ either intersect the interior of the disk D_i, or coincide with an arc of D_i.

Nodes $s = a_1$ and $t = a_{k-j+1}$ are ring points, so we let $v_{\ell_1} = s$ and $v_{\ell_{k-j+1}} = t$. For the remaining points a_i, we let v_{ℓ_i} be the closest vertex vicinity center or ring point to a_i in disk D_i, see Figure 8. Consider now an inter-vertex vicinity portion $SP_w(a_i, a_{i+1})$. We define the path $\pi'(v_{\ell_i}, v_{\ell_{i+1}})$ as the path from v_{ℓ_i} to $v_{\ell_{i+1}}$ through b_{2i}, the point where $\pi(a_i, a_{i+1})$ leaves D_i. Using the triangle inequality, we get that

$$\|\pi'(v_{\ell_i}, v_{\ell_{i+1}})\| \leq \|v_{\ell_i} a_i\| + \|\pi(a_i, a_{i+1})\| + \|a_{i+1} v_{\ell_{i+1}}\|.$$

Moreover, the maximum distance between consecutive Steiner points on the same disk is given by the last two points on the same annulus. This, together with the fact that a_i belongs to the interior of the annulus of some vertex vicinity center, and Equation (2), gives us

$$\|v_{\ell_i} a_i\| \leq 2R_i \sin\left(\frac{\omega_i \varepsilon}{2a}\left(1 - \frac{2\omega_i\varepsilon}{a\pi}\right)^{\frac{1}{\log_2 \frac{a\pi}{a\pi - 2\varepsilon\omega_i}}}\right) \min\{1, \omega_i\}.$$

An upper bound for $\|a_{i+1} v_{\ell_{i+1}}\|$ is obtained analogously. Thus, we have the following inequality:

$$\|\pi'(v_{\ell_i}, v_{\ell_{i+1}})\| \leq \frac{R_i \omega_i \varepsilon}{a} + \|\pi(a_i, a_{i+1})\| + \frac{R_{i+1}\omega_{i+1}\varepsilon}{a}$$

Note that we are using the fact that $\sin\theta \leq \theta$ when $\theta \geq 0$, and $(1-x)^y \leq 1$ when $x, y \geq 0$. Also, the inequality is true even in the case where $\omega_i = 0$ (resp., $\omega_{i+1} = 0$), since $\|v_{\ell_i} a_i\| = 0 \leq \frac{R_i\omega_i\varepsilon}{a}$ (resp., $\|a_{i+1} v_{\ell_{i+1}}\| = 0 \leq \frac{R_{i+1}\omega_{i+1}\varepsilon}{a}$). Now, recall that $a = \frac{1+3c+\sqrt{9c^2+10c+1}}{2} > 1$ since $c = \frac{\frac{\pi}{2}\max_{1\leq j\leq n}\{R_j\}}{\min_{1\leq j\leq n}\{d_j\}} > 0$. Thus, a can be written as the solution of the system of equations given by $a = \frac{bc}{2}$ and $b = \frac{6a+2}{a-1}$. Then,

$$\begin{aligned}&\frac{R_i\omega_i\varepsilon}{a} + \|\pi(a_i, a_{i+1})\| + \frac{R_{i+1}\omega_{i+1}\varepsilon}{a} \\ &= \frac{2\omega_i\varepsilon R_i \min_{1\leq j\leq n}\{d_j\}}{b\frac{\pi}{2}\max_{1\leq j\leq n}\{R_j\}} + \|\pi(a_i, a_{i+1})\| \\ &\quad + \frac{2\omega_{i+1}\varepsilon R_{i+1}\min_{1\leq j\leq n}\{d_j\}}{b\frac{\pi}{2}\max_{1\leq j\leq n}\{R_j\}} \\ &\leq \frac{2\varepsilon d_i}{b} + \|\pi(a_i, a_{i+1})\| + \frac{2\varepsilon d_{i+1}}{b}. \qquad (4)\end{aligned}$$

Therefore, we obtain the path $\pi'(s,t) = \pi'(s, v_{\ell_2}) \cup \pi'(v_{\ell_2}, v_{\ell_3}) \cup \ldots \cup \pi'(v_{\ell_{k-j}}, t)$. For each $i = 1, \ldots, k-j$, we define the point p_i to be the closest Steiner point to b_{2i} that is to the right of $SP_w(s,t)$ when oriented from s to t if b_{2i} is to the right of the segment from a_i to its diametrically opposed point on D_i. Otherwise, we let p_i be the closest Steiner point to b_{2i} that is to the left of $SP_w(s,t)$. Now, we create the path $\pi''(s,t) = \pi''(s, v_{\ell_2}) \cup \pi''(v_{\ell_2}, v_{\ell_3}) \cup \cdots \cup \pi''(v_{\ell_{k-j}}, t)$, where $\pi''(v_{\ell_i}, v_{\ell_{i+1}}) = (v_{\ell_i}, p_i, v_{\ell_{i+1}})$. We know from Lemma 5 that $\|\pi''(v_{\ell_i}, v_{\ell_{i+1}})\| \leq \left(1 + \frac{\varepsilon}{a}\right) \cdot \|\pi'(v_{\ell_i}, v_{\ell_{i+1}})\|$. Thus, using Equation (4),

$$\begin{aligned}\|\pi''(s,t)\| &= \sum_{i=1}^{k-j} \|\pi''(v_{\ell_i}, v_{\ell_{i+1}})\| \\ &\leq \left(1 + \frac{\varepsilon}{a}\right) \sum_{i=1}^{k-j} \|\pi'(v_{\ell_i}, v_{\ell_{i+1}})\| \\ &\leq \left(1 + \frac{\varepsilon}{a}\right) \sum_{i=1}^{k-j} \left(\|\pi(a_i, a_{i+1})\| + \frac{2\varepsilon}{b}(d_i + d_{i+1})\right) \qquad (5) \\ &\leq \left(1 + \frac{\varepsilon}{a}\right) \sum_{i=1}^{k-j} \|\pi(a_i, a_{i+1})\| \\ &\quad + \left(1 + \frac{1}{a}\right) \frac{2\varepsilon}{b} \sum_{i=1}^{k-j} (d_i + d_{i+1}). \qquad (6)\end{aligned}$$

Recall that d_i is the minimum distance from disk D_i to any other disk D_j. Hence, it follows that

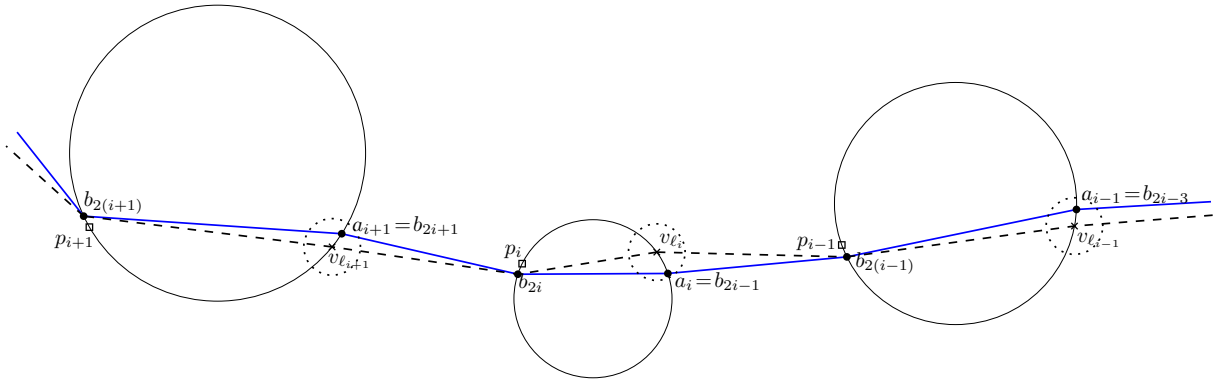


Figure 8: The shortest path $SP_w(s,t)$ is represented in blue. The path $\pi'(s,t)$ is represented as a dashed path. The vertex vicinities are the small disks around $v_{\ell_{i-1}}$, v_{ℓ_i} and $v_{\ell_{i+1}}$.

$$\begin{aligned} d_i + d_{i+1} &\leq (\|v_{\ell_i}a_i\| + \|\pi(a_i, a_{i+1})\|) \\ &\quad + \left(\|v_{\ell_{i+1}}a_{i+1}\| + \|\pi(a_{i+1}, a_i)\|\right) \\ &\leq 2\|\pi(a_i, a_{i+1})\| + \frac{2}{b}(d_i + d_{i+1}). \end{aligned}$$

The second inequality in the previous equation comes from the fact that $\|v_{\ell_i}a_i\| + \|v_{\ell_{i+1}}a_{i+1}\| \leq \frac{2\varepsilon}{b}(d_i + d_{i+1}) \leq \frac{2}{b}(d_i + d_{i+1})$, see Equation (5). Hence, $d_i + d_{i+1} \leq \frac{2b}{b-2}\|\pi(a_i, a_{i+1})\|$. This, when substituted in Equation (6) implies that

$$\begin{aligned} \|\pi''(s,t)\| &\leq \left(1 + \frac{\frac{6a+2}{a-1}\varepsilon + 4a\varepsilon + 2\varepsilon}{a\left(\frac{6a+2}{a-1} - 2\right)}\right) \sum_{i=1}^{k-j} \|\pi(a_i, a_{i+1})\| \\ &= (1+\varepsilon) \cdot \|SP_w(s,t)\|. \end{aligned}$$

Finally, since the length of the shortest path $\tilde{\pi}(s,t)$ in G_ε is at most as large as the length of $\pi''(s,t)$, we obtain the desired result. □

If we use this discretization scheme for disks, the approximation factor that we achieve is better than approximating each disk by a c-gon, and then using the existing methods for triangulations by a factor of approximately $\sqrt{1+\varepsilon}\log_2\frac{2}{\varepsilon}$. The reason might be because most of the discretization schemes we are aware of (see, e.g., [1, 2, 3, 30]) are described in terms of a triangulation and, in our case, we would only have disjoint c-gons.

Remark 7 *Using our discretization scheme for weighted disks provides an approximate shortest path using fewer Steiner points than when using other schemes for triangulations.*

Proof. First, let us consider that the disks are approximated by regular c-gons circumscribing the disks. We would like to know the value of c for which the length of a path that coincides with the boundary of the c-gon is a $(1+\varepsilon)$-approximation of the length of a path that coincides with an arc of a disk. For the rest of the proof we assume that the c-gons are disjoint. Let a be a corner of the c-gon circumscribing D_i, let d be the intersection between the segment $\overline{c_i a}$ and D_i. Let b be the midpoint of an edge containing a, see Figure 9. Now, we want to compare the length of the segment $\overline{ab}$ with the length of the arc $\widehat{db}$. Let α be the angle $\angle c_i b d$, and assume, w.l.o.g., that the c-gons have side length 2. Then, $\widehat{db} = \frac{\alpha}{\tan\alpha} = \frac{\pi}{c}\cot\frac{\pi}{c}$. Hence,

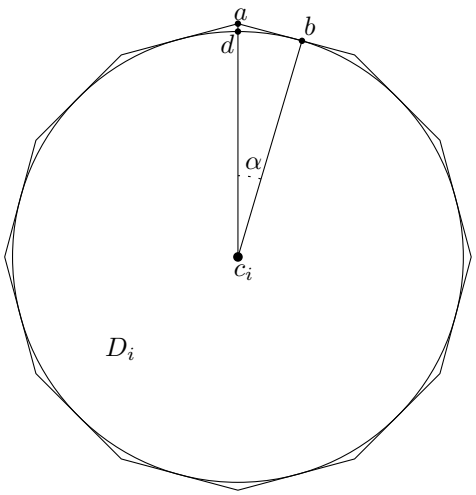


Figure 9: c-gon circumscribing a disk.

$$\begin{aligned} \frac{|ab|}{|\widehat{bd}|} &= \frac{1}{\frac{\pi}{c}\cot\frac{\pi}{c}} = \frac{\sin\frac{\pi}{c}}{\frac{\pi}{c}\cos\frac{\pi}{c}} \leq \frac{\frac{\pi}{c}}{\frac{\pi}{c}\left(1 - \frac{\left(\frac{\pi}{c}\right)^2}{2}\right)} \\ &= \frac{1}{\frac{2-\left(\frac{\pi}{c}\right)^2}{2}} = \frac{2}{2-\left(\frac{\pi}{c}\right)^2}, \end{aligned}$$

and if $c \geq \sqrt{\frac{1+\varepsilon}{2\varepsilon}}\pi$, then $\frac{|ab|}{|\widehat{bd}|} \leq 1+\varepsilon$. The lowest upper bound on the number of vertices of a discretization

scheme is obtained in [3], giving in this case at most $\frac{C(P)\sqrt{\frac{1+\varepsilon}{2\varepsilon}}\pi n \log_2 \frac{2}{\varepsilon}}{\sqrt{\varepsilon}}$ Steiner points, for some parameter $C(P) > 0$. However, using our approach, we are adding at most $C(\mathfrak{D})\frac{n}{\varepsilon}$, for some other parameter $C(\mathfrak{D}) > 0$. Thus,

$$\frac{\frac{C(P)\sqrt{\frac{1+\varepsilon}{2\varepsilon}}\pi n \log_2 \frac{2}{\varepsilon}}{\sqrt{\varepsilon}}}{C(\mathfrak{D})\frac{n}{\varepsilon}} \geq \frac{C(P)\pi}{C(\mathfrak{D})\sqrt{2}}\sqrt{1+\varepsilon}\log_2 \frac{2}{\varepsilon} \qquad (7)$$

We know that $\sqrt{1+\varepsilon}\log_2 \frac{2}{\varepsilon} > 1$, since $\varepsilon > 0$, and that $\frac{C(P)\pi}{C(\mathfrak{D})\sqrt{2}} > 0$, so the value in Equation (7) is at least 1 for small values of ε. This concludes the proof that we are adding less points than if we approximate each disk with a c-gon, and then we use existing algorithms that work on polygons. □

5 Conclusions and open problems

We presented and analyzed a discretization scheme of the 2D space containing a set of non-overlapping weighted disks. Using this scheme, one can compute an approximate shortest path when the disks on the space have a non-negative weight assigned to them. The main idea of the discretization is to place Steiner points on the boundary of the disks.

In addition, we can solve exactly the special case where the disks have a weight $\omega = 0$ or $\omega \geq \frac{\pi}{2}$ by using visibility graph techniques and Dijkstra's algorithm in $O(n^2)$ time. We can also show how to create a linear-sized t-spanner to reduce the running time of the algorithms that compute a weighted shortest path when the disks have any non-negative weight assigned to them. However, due to lack of space, we cannot include these results here.

As future work, it would be interesting to reduce the number of Steiner points that we place on the boundary of the disks, or reduce the number of edges of the associated graph. Finally, a more general version of the problem is to consider some disks that are not mutually disjoint.

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. Approximation algorithms for geometric shortest path problems. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 286–295, 2000.

[3] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM*, 52(1):25–53, 2005.

[4] P. Bose, P. Carmi, and M. Couture. Spanners of additively weighted point sets. *Journal of Discrete Algorithms*, 9(3):287–298, 2011.

[5] P. Bose, G. Esteban, and A. Maheshwari. A Steiner-point-based algorithm for approximate shortest paths in weighted equilateral-triangle meshes. *Theoretical Computer Science*, page 114583, 2024.

[6] I. C. Braid. The synthesis of solids bounded by many faces. *Communications of the ACM*, 18(4):209–216, 1975.

[7] E.-C. Chang, S. W. Choi, D. Kwon, H. Park, and C. K. Yap. Shortest path amidst disc obstacles is computable. In *Proceedings of the Twenty-First Annual Symposium on Computational Geometry*, pages 116–125, 2005.

[8] D. Z. Chen, J. Hershberger, and H. Wang. Computing shortest paths amid convex pseudodisks. *SIAM Journal on Computing*, 42(3):1158–1184, 2013.

[9] D. Z. Chen and H. Wang. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms*, 11(4):1–46, 2015.

[10] L. P. Chew. Planning the shortest path for a disc in $O(n^2 \log n)$ time. In *Proceedings of the First Annual Symposium on Computational Geometry*, pages 214–220, 1985.

[11] M. L. Connolly. An application of algebraic topology to solid modeling in molecular biology. *The Visual Computer*, 3:72–81, 1987.

[12] 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.

[13] L. de Floriani, P. Magillo, and E. Puppo. Applications of computational geometry to geographic information systems. *Handbook of Computational Geometry*, 7:333–388, 2000.

[14] D. P. Dobkin and D. L. Souvaine. Computational geometry in a curved world. *Algorithmica*, 5(1-4):421–457, 1990.

[15] A. Forrest. Invited talk on computational geometry and software engineering. In *ACM Symposium on Computational Geometry*, 1986.

[16] D. Gaw and A. Meystel. Minimum-time navigation of an unmanned mobile robot in a 2-1/2D world with obstacles. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1670–1677. IEEE, 1986.

[17] J. Hershberger, S. Suri, and H. Yildiz. A near-optimal algorithm for shortest paths among curved obstacles in the plane. *SIAM Journal on Computing*, 51(4):1296–1340, 2022.

[18] A. Kamphuis, M. Rook, and M. H. Overmars. Tactical path finding in urban environments. In *First International Workshop on Crowd Simulation*. Citeseer, 2005.

[19] D. G. Kirkpatrick and P. Liu. Characterizing minimum-length coordinated motions for two discs. In *Proceedings of the 28th Canadian Conference on Computational Geometry*, pages 252–259, 2016.

[20] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991.

[21] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman, J. O'Rourke, and C. D. Toth, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 811–848. Chapman and Hall/CRC, 2017.

[22] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[23] T. Pavlidis. Curve fitting with conic splines. *ACM Transactions on Graphics*, 2(1):1–31, 1983.

[24] V. Pratt. Techniques for conic splines. *ACM SIGGRAPH Computer Graphics*, 19(3):151–160, 1985.

[25] N. C. Rowe and R. S. Ross. Optimal grid-free path planning across arbitrarily contoured terrain with anisotropic friction and gravity effects. *IEEE Transactions on Robotics and Automation*, 6(5):540–553, 1990.

[26] M. Sharir and S. Sifrony. Coordinated motion planning for two independent robots. *Annals of Mathematics and Artificial Intelligence*, 3(1):107–130, 1991.

[27] M. Smid. An improved construction for spanners of disks. *Computational Geometry*, 92:101682, 2021.

[28] A. Smith. Invited talk on the complexity of images in the movies. In *ACM Symposium on Computational Geometry*, 1986.

[29] N. R. Sturtevant, D. Sigurdson, B. Taylor, and T. Gibson. Pathfinding and abstraction with dynamic terrain costs. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 80–86, 2019.

[30] Z. Sun and J. H. Reif. On finding approximate optimal paths in weighted regions. *Journal of Algorithms*, 58(1):1–32, 2006.

Appendix

This Appendix is devoted to provide the proof on the size of the graph G_ε.

Lemma 8 *The number of nodes in G_ε is at most $C(\mathfrak{D})\frac{n}{\varepsilon}$, where $C(\mathfrak{D}) < \frac{2a\pi^3(\max_{1\le j\le n}\{R_j\}+1)}{\min\{1,\omega^2\}\cdot\min_{1\le j\le n}\{1,d_j^2,R_j^2\}}$, and ω is the minimum positive weight of $\mathfrak{D}$.*

Proof. Proposition 4 gives us an upper bound on the number of vertex vicinity centers and ring points in each disk with weight greater than 0, which is $\frac{1}{\log_2 \frac{a\pi}{a\pi-2\omega_i\varepsilon}}\frac{\pi}{\alpha_i}$. We know that $\log_2 \frac{1}{1-x} \ge x$, when $x \ge 0$, so

$$\begin{aligned}
\frac{1}{\log_2 \frac{a\pi}{a\pi-2\omega_i\varepsilon}}\frac{\pi}{\alpha_i} &= \frac{\pi}{\alpha_i \log_2 \frac{1}{1-\frac{2\omega_i\varepsilon}{a\pi}}} \le \frac{\pi}{\alpha_i \frac{2\omega_i\varepsilon}{a\pi}} \\
&\le \frac{\pi}{\arcsin\left(\frac{\min\{d_i,R_i\}\min\{1,\omega_i\}}{4R_i\max\{1,\omega_i\}}\right)\frac{2\omega_i\varepsilon}{a\pi}} \\
&\le \frac{\pi}{\left(\frac{\min\{d_i,R_i\}\min\{1,\omega_i\}}{4R_i\max\{1,\omega_i\}}\right)\frac{2\omega_i\varepsilon}{a\pi}} \\
&= \frac{2a\pi^2 R_i\max\{1,\omega_i\}}{\min\{d_i,R_i\}\min\{1,\omega_i\}\omega_i\varepsilon} \\
&\le \frac{2a\pi^2(R_i+1)\frac{\pi}{2}}{\min\{d_i,R_i\}\min\{1,\omega_i\}\omega_i\varepsilon} \\
&< \frac{2a\pi^3(R_i+1)}{\min\{d_i,R_i\}\min\{\omega_i,\omega_i^2\}\varepsilon} \\
&\le \frac{2a\pi^3(R_i+1)}{\min\{d_i,R_i\}\min\{1,\omega_i^2\}\varepsilon}.
\end{aligned}$$

Moreover, if the weight of D_i is 0, then by Equation (1) we are placing $\frac{2\pi}{\frac{\varepsilon d_i}{a(d_i+1)}R_i}$ points, so

$$\begin{aligned}
\frac{2\pi}{\frac{\varepsilon d_i}{a(d_i+1)}R_i} = \frac{2a\pi(d_i+1)}{R_i\varepsilon d_i} &= \begin{cases} \overbrace{=}^{\text{if } R_i\le d_i} \frac{2a\pi(R_i+1)}{R_i^2\varepsilon} \\ \overbrace{=}^{\text{if } d_i<R_i} \frac{2a\pi(d_i+1)}{d_i^2\varepsilon} \end{cases} \\
&= \frac{2a\pi(\min\{d_i,R_i\}+1)}{\min\{d_i^2,R_i^2\}\varepsilon} \le \frac{2a\pi(R_i+1)}{\min\{d_i^2,R_i^2\}\varepsilon} \\
&< \frac{2a\pi^3(R_i+1)}{\min\{d_i^2,R_i^2\}\varepsilon}.
\end{aligned}$$

Then, since we have n disks, the total number of vertex vicinity centers and ring points is upper-bounded by $C(\mathfrak{D})\frac{n}{\varepsilon}$, where $C(\mathfrak{D}) < \frac{2a\pi^3(\max_{1\le j\le n}\{R_j\}+1)}{\min\{1,\omega^2\}\cdot\min_{1\le j\le n}\{1,d_j^2,R_j^2\}}$, where ω is the minimum positive weight of $\mathfrak{D}$. Thus, the estimate on the number of nodes in G_ε is $O(\frac{n}{\varepsilon})$. Note that here we are taking into account that a does not depend on ε.

The set E_ε of edges is obtained by creating an edge (u,v) between any two vertex vicinity centers or ring points. In case u and v are adjacent on the disk, we add an arc between them. In addition, if u and v are not visible, the edge is a non-straight line segment. This edge is a shortest path from u to v avoiding all the disks, see, e.g., the red path in Figure 6. Thus, the total number of edges in G_ε is $O(\frac{n^2}{\varepsilon^2})$. □

Burning Simple Polygons

Justin Bruss* William Evans Jiaxuan Li

Abstract

Given a simple polygon P with n vertices and an integer k, we wish to find a set S of k vertices of P that minimize the maximum geodesic distance from any point in P to its geodesically closest vertex in S. We describe a dynamic programming algorithm that solves this problem in $O(n^7)$ time.

1 Introduction

We are given a simple polygon P defined by n vertices V, ordered clockwise around P. We are also given a positive integer $k \leq n$. The problem is to find a set U of k vertices in V that minimizes (over all size k subsets of V) the maximum geodesic distance (over all points $p \in P$) from U to p, where the distance from U to p is $d(U,p) \equiv \min_{u \in U} d(u,p)$ and $d(u,p)$ is the geodesic distance (length of the shortest path in P) from u to p. Alternatively, one can think of the set $U \subseteq V$ as being *ignition* points where we simultaneously ignite k fires that totally burn the flammable polygon in the least time.

Our problem is a restricted version (where $U \subseteq V$) of the general geodesic k-center problem in a simple polygon P where $U \subseteq P$. Oh et al. [7] described an $O(n^2 \log^2 n)$-time algorithm for the general problem in a simple polygon for $k = 2$. Soon after, Cho et al. [3] announced an optimal $O(n \log n)$ algorithm for this problem. The general problem for $k > 2$ seems more difficult. Evans and Lin [4] introduced the polygon burning problem and showed it to be NP-hard for polygons with holes. They also gave an algorithm that solves the problem in $O(kn^2)$ time for a restricted class of convex polygons. In this paper, we describe an algorithm to solve the problem for simple polygons in $O(n^7)$ time.

Our approach starts by identifying a small set of points in P, called the *potential final burn points*, denoted $\mathcal{F}$, that contains all points that could be the last to burn in P for any k and any $U \subseteq V$. The set $\mathcal{F}$ comprises precisely the vertices V, along with the points on the boundary of P equidistant from two vertices of P, and the points within P equidistant from three vertices of P. Thus $\mathcal{F}$ consists of points defined by one, two, or three vertices of P. Consequently, $|\mathcal{F}| \in O(n^3)$ and it can be calculated in $O(n^4)$ time [6]. For a point $p \in \mathcal{F}$ to be an actual final burn point of a set of $U \subseteq V$ of k ignition points, the vertex/vertices that define it must be in U and there must be a way to choose the remaining ignition points in U so that (1) U contains no vertex of P closer to p than p's distance, $t(p)$, to its defining points[1] (i.e. p is the last point to burn) and (2) every point in P is at distance at most $t(p)$ to its closest point in U (i.e. all of P burns). Given a particular value k, our algorithm considers each $p \in \mathcal{F}$, in increasing order of $t(p)$, as a candidate for the true final burn point for some set U of k ignition points. Using dynamic programming, our algorithm determines if such a set U exists. The first such p is the geodesic k-center.

2 Properties and Definitions

Notation: Let P be a simple polygon with vertices V. Let ∂P be the boundary of P and let $\partial P(a,b), a, b \in \partial P$ be the boundary of P clockwise from a up to b. For $u, v \in P$, let $\pi(u,v)$ be the shortest geodesic path in P from u to v and let $d(u,v)$ be the length of $\pi(u,v)$. The last vertex (or u if there is none) before v on $\pi(u,v)$ is called the *anchor* of v (with respect to u). Let $b(u,v)$ be the geodesic bisector of u and v in P. Given a set of ignition points (sites) $U \subseteq V$, let $\texttt{Vor}(U)$ be the geodesic nearest-point Voronoi diagram of U in P and let $\texttt{Vor}(U)[u]$ be the Voronoi region associated with $u \in U$ in $\texttt{Vor}(U)$. We also call $\texttt{Vor}(U)[u]$ the *burn region* associated with u.

Observation 1 *Since we are considering burn regions which are Voronoi regions of sites that are vertices of P, every burn region contains a vertex of P. In fact, burn regions are star-shaped[2] around their ignition point (from Aronov [1, Cor. 3.20]).*

Definition 1 (Burn Path) *A burn path $\pi(u,p)$ is the shortest geodesic path from an ignition point u to a point p in the burn region associated with u.*

Definition 2 (Final Burn Point) *A final burn point of a set of ignition points $U \subseteq V$ is a point $p \in P$*

*Department of Computer Science, University of British Columbia, justinbruss1@gmail.com will@cs.ubc.ca jack.li.jxl@gmail.com. Research supported in part by NSERC Discovery Grant.

[1] A point p may have different values of $t(p)$ if it has different sets of defining points.

[2] A set $Q \subset P$ is *star-shaped* around $p \in P$ (with respect to the geodesic metric) if $\forall q \in Q : \pi(p,q) \subset Q$.

satisfying

$$d(U,p) = \sup_{q \in P} d(U,q)$$

For simplicity, we will assume that P is in *general position*, meaning no vertex of P is equidistant from two other vertices of P, i.e., no bisector $b(u,v)$, $u,v \in V$ contains a vertex of P (from [1, Def. 3.21]). However, this assumption can be removed (See Appendix A). Note that P being in general position implies that all bisectors intersect the boundary of P at exactly two points.

Lemma 1 *Let $u,v \in V$. Then $d(u,p)$ where $p \in b(u,v)$ is strictly convex in p.*

Proof. See Appendix B for proof. □

Corollary 2 *Given the open subset S of $b(u,v)$ between two points p and q in $b(u,v)$*

$$\max\{d(u,p), d(u,q)\} > \sup_{p' \in S} d(u,p')$$

Proof. Since $d(u,p')$ is strictly convex in $p' \in b(u,v)$, the maximum value of $d(u,p')$ is attained at the boundaries of a given subdomain. □

Lemma 3 *Given a burn region R associated with ignition point u, there exists $p \in R$ such that $d(u,p) = \sup_{p' \in R} d(u,p')$. Moreover, for all such $p \in R$, at least one of the following is true:*

1. *$p \in (V \cap R) \setminus \{u\}$*
2. *p is the intersection of $b(u,v)$ with ∂P for some $v \in V$*
3. *p is the intersection of $b(u,v)$ with $b(u,w)$ for some $v,w \in V$*

Proof. See Appendix C for proof. □

Definition 3 (Potential Final Burn Point) *A potential final burn point is a point $p \in P$ such that:*

- *$p \in \bigcap_{n \in N} \texttt{Vor}(N)[n]$ for some $N \subseteq V$ where $1 \leq |N| \leq 3$.*
- *If $N = \{u\}$, $p \in V \setminus \{u\}$.*
- *If $N = \{u,v\}$, p is the intersection of $b(u,v)$ with ∂P.*
- *If $N = \{u,v,w\}$, p is the intersection of $b(u,v)$ with $b(u,w)$. (This condition, when $|N| = 3$, is redundant.)*

We use $\mathcal{F}$ to denote the set containing all such potential final burn points in P. For $p \in \mathcal{F}$, we call its corresponding N the set of defining ignition points of p.

Classifying points $p \in \mathcal{F}$ by the number of ignition points used to define them, we obtain a constructive definition of $\mathcal{F}$:

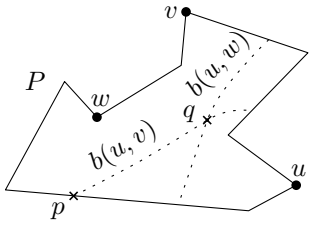


Figure 1: p is the Type II potential final burn point defined by u,v clockwise from u to v. q is the Type III potential final burn point defined by u,v,w.

A **Type I** potential final burn point is defined by a single ignition point ($|N| = 1$). All n vertices are Type I potential final burn points.

Type II: A Type II potential final burn point is defined by a pair of ignition points $N = \{u,v\}$. There are $\binom{n}{2}$ ways to choose distinct $u,v \in V$. For each pair, the bisector $b(u,v)$ intersects ∂P, yielding exactly two Type II potential final burn points (Fig. 1). Thus, we have up to $2\binom{n}{2}$ Type II potential final burn points.

Type III: A Type III potential final burn point is defined by a triple of ignition points $N = \{u,v,w\}$. There are $\binom{n}{3}$ ways to choose distinct $u,v,w \in V$. For each triple, if the corresponding Voronoi vertex exists in P, it constitutes a Type III potential final burn point (Fig. 1). (Note that potential final burn points can have multiple valid sets of defining ignition points.)

It is clear that $\mathcal{F}$ comprises precisely the vertices V, along with the points on the boundary of P equidistant from two vertices of P, and the points within P equidistant from three vertices of P.

Lemma 4 *$\mathcal{F}^* \subseteq \mathcal{F}$ where $\mathcal{F}^*$ is the set of final burn points of P for every possible set of ignition points $U \subseteq V$.*

Proof. Concretely,

$$\mathcal{F}^* = \bigcup_{U \subseteq V} \{p \in P | d(U,p) = \max_{q \in P} d(U,q)\}.$$

Fix $U \subseteq V$ and let $a \in \{p \in P | d(U,p) = \max_{q \in P} d(U,q)\}$. We must have $a \in \texttt{Vor}(U)[u]$ for some $u \in U$. Let R denote the burn region associated with u, $R = \texttt{Vor}(U)[u]$.

Let $b \in R$,

$$\begin{aligned}
d(u,b) &= d(U,b) && b \in \texttt{Vor}(U)[u] \\
&\leq \max_{q \in R} d(U,q) && b \in R \\
&\leq \max_{q \in P} d(U,q) && R \subseteq P \\
&= d(U,a) && \text{choice of } a \\
&= d(u,a) && a \in \texttt{Vor}(U)[u]
\end{aligned}$$

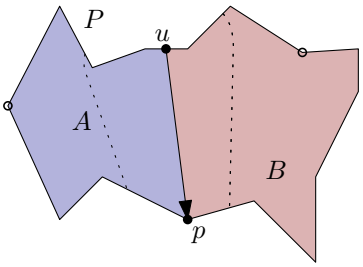


Figure 2: Enclosed regions $A = \mathcal{R}(\{u\}, p, (p, u))$ and $B = \mathcal{R}(\{u\}, p, (u, p))$ defined by ignition point u and a point p which lies on the boundary of P in $\texttt{Vor}(U)[u]$ $(U \subseteq V)$.

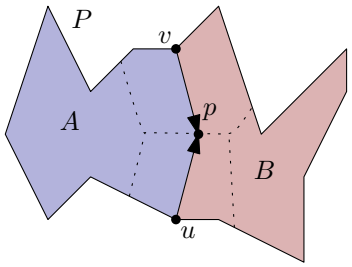


Figure 3: Enclosed regions $A = \mathcal{R}(\{u, v\}, p, (u, v))$ and $B = \mathcal{R}(\{u, v\}, p, (v, u))$ A and B defined by ignition points u, v and a point p which lies in the intersection of the burn regions associated with u and v in $\texttt{Vor}(U)$ $(U \subseteq V)$.

Therefore $d(u, a) = \sup_{b \in R} d(u, b)$ and a satisfies at least one of the three statements in Lemma 3.

If $a \in (V \cap R) \setminus \{u\}$, then we take $N = \{u\}$. If a is the intersection of $b(u, v)$ with ∂P for some $v \in V$, then we take $N = \{u, v\}$. If a is the intersection of $b(u, v)$ with $b(u, w)$ for some $v, w \in V$, then we take $N = \{u, v, w\}$. In all three cases it is clear that a satisfies Definition 3 and is therefore in $\mathcal{F}$.

Since each set in the union is a subset of $\mathcal{F}$, we have $\mathcal{F}^* \subseteq \mathcal{F}$ as desired. □

Note that for all final burn points $p \in \mathcal{F}^*$, the sets of defining points are covered by the sets of defining points for the same $p \in \mathcal{F}$ since we use every possible set.

2.1 Enclosed Regions

Given a set of ignition points $U \subseteq V$, an *enclosed region* is a region of P that is enclosed by a simple curve composed of one or two shortest paths from ignition points in U to some (shared) point p in their associated burn regions, and a continuous part of the boundary of P between the two end points of the curve. More specifically: Let $u \in U$ be an ignition point and $p \in \partial P$ be a point in $\texttt{Vor}(U)[u]$. The region enclosed by $\partial P(u, p)$ (or $\partial P(p, u)$) and $\pi(u, p)$ is an enclosed region with enclosing vertex u and enclosing point p (Fig. 2). Similarly, let $u, v \in U$ be ignition points such that $\texttt{Vor}(U)[u]$ and $\texttt{Vor}(U)[v]$ intersect at some point $p \in P$. The region enclosed by $\partial P(u, v)$ (or $\partial P(v, u)$) and the shortest paths $\pi(u, p)$ and $\pi(v, p)$ is an enclosed region with enclosing vertices u, v and enclosing point p (Fig. 3).

Let $\mathcal{R}(N, p, (a, b))$ denote the enclosed region of P enclosed by $\pi(N, p) \cup \partial P(a, b)$ where N is a set of one or two enclosing vertices, p is the enclosing point associated with the vertices, and $a, b \in \partial P$ are the end points of $\pi(N, p)$. Note that the ordered pair (a, b), with $a, b \in N \cup \{p\}$, determines which half of P is enclosed.

Lemma 5 (Isolation Property) *Given an enclosed region $R = \mathcal{R}(N, p, (a, b))$ where the paths $\pi(N, p)$ are burn paths in $\texttt{Vor}(U)$, no region in $\texttt{Vor}(U)$ exists on both sides of $\bigcup_{v \in N} \pi(v, p)$ (other than the regions associated with N).*

Proof. If the paths $\pi(N, p)$ are burn paths in $\texttt{Vor}(U)$, $\texttt{Vor}(U)$ contains regions associated with the ignition points in N. Furthermore, $\bigcup_{v \in N} \pi(v, p)$ is entirely contained in $\bigcup_{v \in N} \texttt{Vor}(U)[v]$ and must separate P into two halves. By Lemma 1, burn regions are simply connected. If there exists a region in $\texttt{Vor}(U)$ that is on both sides of $\bigcup_{v \in N} \pi(v, p)$ that isn't associated with vertices in N, it would imply it is not simply connected or that the region intersects other regions at more than just the boundary which contradicts our general position assumption. □

3 Algorithm Description

The input is a simple polygon P with vertices V, and the maximum number of vertex ignition points $k > 0$ allowed to burn P.

Definition 4 (Properly Burned) *Given a set of ignition points $U \subseteq V$, an enclosed region R is considered* properly burned *in time t if the burn paths enclosing R exist in $\texttt{Vor}(U)$ and R is entirely burned in time t.*

3.1 Preprocessing

We first calculate and store the set of all Type II and Type III potential final burn points (see Definition 3). Let $\texttt{F}_2[u, v]$ be the Type II potential final burn point $p \in \partial P(u, v)$, if it exists, with associated ignition points $u, v \in V$. Let $\texttt{F}_3[u, v, w]$[3] be the Type III potential final burn point p, if it exists, with associated ignition points $u, v, w \in V$. Populating these tables takes time $O(n^4)$ using the algorithm of Oh [6] which takes $O(n)$ time to compute the geodesic Voronoi diagram for a constant number of sites. It takes time $O(n^4 \log n)$ using the simpler algorithm of Asano and Asano [2].

[3] Assume different permutations of u, v, w index the same way.

3.2 Polygon Burning Algorithm

The Polygon Burning Algorithm returns the solution to the Polygon Burning Problem in P by finding a set of ignition points $U \subseteq V$, $|U| \leq k$, such that the time taken to burn the entirety of P is minimized. This is done by iterating through every point $p_0 \in \mathcal{F}$ in increasing order of associated final burn time, finding the smallest set U of ignition points that achieves that burn time, until we find U such that $|U| \leq k$.

(1) Let `DPTable` be a dynamic programming table, indexed by a time $t(p_0)$ and an enclosed region R, that stores the minimum size set of ignition points required to properly burn R in time $t(p_0)$.

(2) For each potential final burn point p_0 with associated set of ignition points $N = \{u\}$ (Type I), $N = \{u, v\}$ (Type II), or $N = \{u, v, w\}$ (Type III), in increasing order of $t(p_0) = d(u, p_0)$, we attempt to find the smallest set $U \subseteq V$ of ignition points that burn the entirety of P with final burn point p_0 and the burn paths from N to p_0 appear in $\texttt{Vor}(U)$. There are different initial enclosed regions for each type of potential final burn point:

Type I: There are two enclosed regions associated with a Type I potential final burn point: $\mathcal{R}(\{u\}, p_0, (u, p_0))$, $\mathcal{R}(\{u\}, p_0, (p_0, u))$. These correspond to the halves of the polygon clockwise of u up until p_0 and clockwise of p_0 up until u respectively which are separated by the burn path $\pi(u, p_0)$.

Type II: Without loss of generality, assume p_0 is clockwise of u up to v. There are three enclosed regions associated with a Type II potential final burn point: $\mathcal{R}(\{u\}, p_0, (u, p_0))$, $\mathcal{R}(\{v\}, p_0, (p_0, v))$, $\mathcal{R}(\{u, v\}, p_0, (v, u))$. These correspond to 3 regions of the polygon, one isolated by the burn path $\pi(u, p_0)$, one isolated by the burn path $\pi(v, p_0)$, and one isolated by the burn paths $\pi(u, p_0)$ and $\pi(v, p_0)$.

Type III: There are three enclosed regions associated with a Type III potential final burn point (note that we are assuming that u, v, w are in clockwise order around P): $\mathcal{R}(\{u, v\}, p_0, (u, v))$, $\mathcal{R}(\{v, w\}, p_0, (v, w))$, $\mathcal{R}(\{w, u\}, p_0, (w, u))$. These correspond to 3 regions of the polygon which are each enclosed by two burn paths from ignition points to p_0.

After determining the set of enclosed regions associated with p_0, we attempt to find the smallest set of ignition points as follows:

(a) We determine the sets $S_1, S_2, (S_3)$ corresponding to the optimal set of ignition points required to burn the enclosed regions associated with p_0 and N in time $t(p_0)$ using the ERA algorithm in Section 3.3.

(b) If $S_1 = \emptyset$ or $S_2 = \emptyset$ (or $S_3 = \emptyset$), this means it is not possible for p_0 to be a true final burn point so we continue to the next potential final burn point; otherwise

(c) let $U \leftarrow S_1 \cup S_2 (\cup S_3)$

Once we find U such that $|U| \leq k$, we return U.

Lemma 6 *If it is possible to burn P with a set of ignition points U such that p_0 is the final burn point and the paths from N to p_0 are burn paths in $\texttt{Vor}(U)$ then the algorithm returns the smallest such set. Otherwise it continues to the next potential final burn point.*

Proof. For each potential final burn point p_0 defined by a set of ignition points N, there exist known burn paths from N to p_0. Using these known paths, we define a set of enclosed regions S_R as in Step 2 such that each region in S_R is isolated from the rest by burn paths from N to p_0 and $\bigcup S_R = P$. Let $t(p_0) = d(N, p_0)$ (by definition, each ignition point in N is equidistant to p_0). Properly burning all of the regions in S_R in time $t(p_0)$ with a set of ignition points U is equivalent to P being entirely burnt in time $t(p_0)$, p_0 being a final burn point in P, and burn paths $\pi(N, p_0)$ being burn paths in $\texttt{Vor}(U)$. This is because properly burning all regions in S_R with U implies:

- The known burn paths from N to p_0 are burn paths in $\texttt{Vor}(U)$
- $\bigcup S_R = P$ is entirely burned
- Each region in S_R is burned in time less than or equal to $t(p_0)$ which means p_0, which is burned at exactly time $t(p_0)$, is the last point to burn.

Suppose U^* is a smallest set of ignition points that properly burns all of the regions in S_R. Since each region in S_R is isolated from the rest by known burn paths, U^* must properly burn each region in S_R optimally. By Theorem 9, in Step 2a, we obtain optimal sets for properly burning each region in S_R. Since U^* properly burns each region optimally, the union U of the optimal sets found in Step 2a has the same magnitude as U^* which implies U is optimal.

If it is not possible to burn P and maintain p_0 as the final burn point with burn paths from N to p_0, By Theorem 9, we will not be able to properly burn at least one of the regions in S_R which means we will continue to the next potential final burn point in Step 2b. □

Theorem 7 *This algorithm finds a set of ignition points U, $|U| \leq k$, such that for all points $p \in P$, the maximum geodesic distance from U to p is minimized.*

Proof. By Lemma 4, $\mathcal{F}^*$ is a subset of F_P. In addition, the set of defining points for each $p \in \mathcal{F}^*$ is covered

by the set of sets of defining points for corresponding $p' \in \mathcal{F}$. Suppose U^*, $|U^*| \leq k$, is an optimal set of ignition points that minimizes the maximum distance from an ignition point to a point $p \in P$. U^* will have an associated final burn point $p_0 \in P$. Since $\mathcal{F}$ covers every possible set of initial ignition points that make p the final point to burn in F (Lemma 4), if we look at potential final burn point p_0 with initial ignition points N such that the burn paths from N to p_0 exist as burn paths in $\texttt{Vor}(U^*)$, we will return a set U such that $|U| = |U^*|$ by Lemma 6.

Since U^* is optimal and we explore potential final burn points in increasing order of burn time, if we return before we get to potential final burn point p_0 with initial ignition points N, we must have returned a set U, $|U| \leq k$, with burn time equal to the burn time of U^*. □

3.3 Enclosed Region Algorithm (ERA)

The purpose of this algorithm is to fill in the dynamic programming table by finding the set with the minimum number of ignition points required to properly burn an enclosed region R of P in time $t(p_0)$. If it is not possible to properly burn R in time $t(p_0)$, the table is assigned the empty set.

Each subproblem is defined by an enclosed region $R = \mathcal{R}(N = \{u, (v)\}, p, (a, b))$ and a time $t(p_0)$ to burn R. Note that we assume $d(N, p) \leq t(p_0)$.

(1) Return $\texttt{DPTable}[t(p_0), R]$ if it has already been calculated; otherwise

(2) Determine if R is entirely burned in time $t(p_0)$ by the set N of enclosing vertices in linear time (see Section E). If R is entirely burned, we update the dynamic programming table: $\texttt{DPTable}[t(p_0), R] \leftarrow N$ and return N; otherwise

(3) Let $S_{best} \leftarrow \emptyset$ be the minimum set of ignition points required to burn R.

(4) If $|N| = 2$, attempt to recursively find the optimal sets S_A and S_B for enclosed regions A and B implied by Case 2 of Lemma 12. If S_A and S_B are both non-empty, $S_{best} \leftarrow S_A \cup S_B$.

(5) For each vertex $v' \in V \cap \partial P(a, b)$, we attempt to add v' as the next enclosing vertex.

  (a) Determine the enclosing point p' and enclosing regions A and B associated with adding v' as the next enclosing vertex (see Section 3.4). If we cannot add v' as the next enclosing vertex, we continue to the next iteration of the loop; otherwise

  (b) Recursively find optimal ignition point sets S_A and S_B that properly burn A and B respectively in time $t(p_0)$.

  (c) If $S_A = \emptyset$ or $S_B = \emptyset$, A and/or B could not be properly burned in time $t(p_0)$ so we continue to the next iteration; otherwise

  (d) If $|S_A \cup S_B| < |S_{best}|$ or $S_{best} = \emptyset$, $S_{best} \leftarrow S_A \cup S_B$.

(6) Update the dynamic programming table $\texttt{DPTable}[t(p_0), R] \leftarrow S_{best}$ and return S_{best}.

Lemma 8 *In ERA Step 4, given that burn paths $\pi(u, p)$ and $\pi(v, p)$ exist, properly burning A and B is equivalent to properly burning R, subject to $b(u, v)$ not intersecting any region in R other than $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$.*

Proof. If A and B are properly burned, burn paths $\pi(u, p')$ and $\pi(v, p')$ exist and are burned in time $t(p_0)$. This means burn paths $\pi(u, p)$ and $\pi(v, p)$ are burn paths in $\texttt{Vor}(U)$ since they are separated from $A \cup B$ by $\pi(u, p') \cup \pi(v, p')$. Furthermore, $R_{u,v} = R \backslash (A \cup B)$ which is enclosed by $\pi(u, p')$, $\pi(u, p)$, $\pi(v, p')$, and $\pi(v, p)$ is entirely burned by u and v (Obs. 5). This means $R = A \cup B \cup R_{u,v}$ is entirely burned in time $t(p_0)$ which implies R is properly burned. □

Theorem 9 *ERA finds a minimum set of ignition points to properly burn $R = \mathcal{R}(N = \{u, (v)\}, p, (a, b))$ in time $t(p_0)$, if it is possible. Otherwise, ERA returns the empty set.*

Proof. Suppose there exists some optimal set $U \subseteq V$ of ignition points that properly burn R in time t.

- If R is entirely burned by N in time $t(p_0)$, N is the optimal set since we don't need to add more ignition points and the enclosing vertices and enclosing point of R are preserved by default (Step 2).
- If R is not entirely burned by N in time t, there are three cases:

Case $N = \{u\}$: Without loss of generality, suppose $R = \mathcal{R}(\{u\}, p, (p, u))$. By Lemma 11, there exists an ignition point $v' \in U \cap \partial P(p, u)$ and a point $p' \in b(u, v') \cap \partial P(p, v')$ such that $\partial P(p, p')$ is entirely in $\texttt{Vor}(U)[u]$. Since $U \cap \partial P(a, b) \subseteq V \cap \partial P(a, b)$, at some point, we will attempt to add v' as the next enclosing vertex (ERA step 5). Since the enclosed regions A and B implied by adding v' exist in the context of U, by Theorem 10, we will obtain A and B such that properly burning A and B is equivalent to R being properly burned with v' as the next enclosing vertex (ERA step 5a). We recursively find optimal ignition sets S_A and S_B that properly burn regions A and B. Since A and B are isolated from each other in U by $\pi(v', p')$, they must be burned optimally in U or there would exist a set of ignition points more optimal than U. This implies $|S_A \cup S_B| = |U|$. Since U is optimal, the set $S_A \cup S_B$ of ignition points returned by the algorithm is optimal.

Case $N = \{u, v\}$ and $b(u, v)$ intersects regions other than $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$ in R: Without loss of generality, suppose $R = \mathcal{R}(\{u, v\}, p, (v, u))$. By Lemma 12 Case 1, there exists $v' \in U \cap \partial P(v, u)$ and $p' \in R$ such that p' is the Voronoi vertex of u, v, v' (it is equidistant from all three vertices). In a similar manner to the previous case, we will eventually attempt to add v' as the next enclosing vertex and find the regions A and B implied by adding v' as the next enclosing vertex (Obs. 4). We recursively find the optimal sets S_A and S_B that properly burn A and B. Since A and B are isolated from each other by $\pi(v', p')$, they must be burned optimally in U which implies $|S_A \cup S_B| = |U|$. Since U is optimal, the set $S_A \cup S_B$ of ignition points returned by the algorithm is optimal.

Case $N = \{u, v\}$ and $b(u, v)$ does not intersect any region in R other than $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$: In Step 4, we find the regions A and B implied by this case as shown in Case 2 of Obs. 4. Lemma 8 says that properly burning A and B is equivalent to properly burning R in this case. Since it must be possible to properly burn A and B (since U exists), we recursively find the optimal sets S_A and S_B that properly burn A and B. Since A and B are isolated from each other by $b(u, v)$, they must be burned optimally in U which implies $|S_A \cup S_B| = |U|$. Since U is optimal, the set $S_A \cup S_B$ of ignition points returned by the algorithm is optimal.

If there exists no way to properly burn R with some set $U \subseteq V$, for each v', either there will be no way to find A and B (Theorem 10 for Case 1,2, Lemma 8 for Case 3) or we will not be able to recursively find sets that properly burn A and B which means S_{best} will never be updated. Since S_{best} is never updated, we will fill the table with the empty set. □

3.4 Next Enclosing Vertex Algorithm

Given an enclosed region $R = \mathcal{R}(N, p, (a, b))$, a maximum time $t(p_0)$ allowed to burn R, and a potential next enclosing vertex $v' \in \partial V(a, b)$, we determine if v' can be added as the next enclosing vertex and we find enclosed regions A and B associated with adding v'. We assume that the burn paths enclosing R are already burned in time $t(p_0)$.

(1) If $d(v', p) < d(N, p)$, return $\emptyset$; otherwise

(2) There are two cases:

Case 1: $N = \{u\}$: Without loss of generality, assume R is bounded by $\partial P(p, u)$. We assume v' is the vertex described in Lemma 11 which means $p' = \texttt{F}_2[u, v']$ since p' is the intersection of $b(u, v')$ with ∂P on the same side of ∂P as p.

(a) If $d(v', p') > t(p_0)$, return $\emptyset$; otherwise

(b) If $\partial P(p, p')$ is not covered by u in time $t(p_0)$ (see Section E), return $\emptyset$; otherwise

(c) return A and B as in Obs. 3.

Case 2: $N = \{u, v\}$: Without loss of generality, assume R is bounded by $\partial P(v, u)$. We assume v' is the vertex described in Case 1 of Lemma 12 which means $p' = \texttt{F}_3[u, v, v']$ since it is equidistant from all three vertices (if p' does not exist, return $\emptyset$).

(a) If $d(v', p') > t(p_0)$, return $\emptyset$; otherwise

(b) return A and B as in Obs. 4.

Theorem 10 *Given an enclosed region $R = \mathcal{R}(N, p, (a, b))$, a time $t(p_0)$ and v' as the next enclosing vertex, if it is possible, the Next Enclosing Vertex Algorithm gives us two enclosed regions A and B separated by $\pi(v', p')$, such that A and B being properly burned in time $t(p_0)$ by a set of ignition points U is equivalent to R being properly burned in time $t(p_0)$ and paths $\pi(N, p')$ and $\pi(v', p')$ being burn paths in $\texttt{Vor}(U)$. If it is not possible the Next Enclosing Vertex Algorithm returns nothing.*

Proof. See Section D. □

3.5 Runtime Analysis

There are $O(n^3)$ possible values of t for indexing into the dynamic programming table. At each step of the recursion, we choose an enclosed region such that the enclosing point is a potential final burn point of P. This is true for the Polygon Burning Algorithm and in each case of the Next Enclosing Vertex Algorithm. So there are $O(n^3)$ possible enclosed regions meaning the total size of the dynamic programming table is $O(n^6)$. Each value in the table is filled in by one call to ERA. Each iteration of the loop in ERA takes constant time if we precalculate region coverage for vertices ($O(n^6)$ with brute force) and the set of all pairwise distances between $v \in V$ and $\mathcal{F}$ ($O(n^4 \log n)$ using [5]). Since ERA performs $O(n)$ iterations, the final run time of the algorithm is $O(n^7)$.

4 Open Problems

It seems likely that the running time of this algorithm can be improved. It is also likely that a slightly more complicated algorithm could handle a constant number of ignition points in the interior of polygon P.

References

[1] B. Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1):109–140, Dec. 1989.

[2] T. Asano and T. Asano. Voronoi diagram for points in a simple polygon. In D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, editors, *Discrete Algorithms and Complexity*, pages 51–64. Academic Press, 1987.

[3] K. Cho, E. Oh, H. Wang, and J. Xue. Optimal algorithm for the planar two-center problem. *arXiv preprint arXiv:2007.08784*, 2020.

[4] W. Evans and R. Lin. The polygon burning problem. In *WALCOM: Algorithms and Computation*, pages 123–134, 2022.

[5] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

[6] E. Oh. Optimal algorithm for geodesic nearest-point voronoi diagrams in simple polygons. In *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–409, 2019.

[7] E. Oh, J.-L. De Carufel, and H.-K. Ahn. The geodesic 2-center problem in a simple polygon. *Computational Geometry*, 74:21–37, 2018.

Appendix

A Removing the General Position Assumption

Assuming no vertex of P is equidistant to two other vertices ensures that the (geodesic) bisector of any two vertices of P is a 1D curve that intersects ∂P at exactly two points. Without this general position assumption, we can get positive area intersections of Voronoi cells as shown in Figure 4. To handle this, we define the bisector $b(u, v)$ to be only the 1D curve portion of the intersection between Voronoi cells of u and v in $\texttt{Vor}(U)$ (including the two intersection points on ∂P at the ends of the curve).

Observation 2 *All points in P on the same side of $b(u, v)$ as u are at least as close to u as they are to v*

Proof. If there exists a point p such that $d(u, p) > d(v, p)$ and $\pi(v, p)$ crosses $b(u, v)$ at a point x where $d(u, x) = d(v, x)$ then $d(u, p) \leq d(u, x) + d(x.p) = d(v, x) + d(x, p) = d(v, p)$ is a contradiction. □

Assume $b(u, v)$ intersects ∂P at some vertex w. If this bisector appears when running ERA, it must be the case that we are attempting to include both u and v in our solution which means we can assume any non-trivial intersection region is covered by the ignition point on the same side of $b(u, v)$ as the intersection region by Obs. 2.

B Proof of Lemma 1

Lemma 1 *Let $u, v \in V$. Then $d(u, p)$ where $p \in b(u, v)$ is strictly convex in p.*

Proof. The tangent to $b(u, v)$ bisects the angle between the directions to u and v of the shortest paths $\pi(p, u)$ and $\pi(p, v)$[4]. Let u' and v' be the anchor vertices along $\pi(p, u)$ and $\pi(p, v)$ respectively. Let the angle θ be the angle between $\overline{pu'}$ and $\overline{pv'}$ clockwise from $\overline{pv'}$ to $\overline{pu'}$.

[4] from [1, Lemma 3.22]

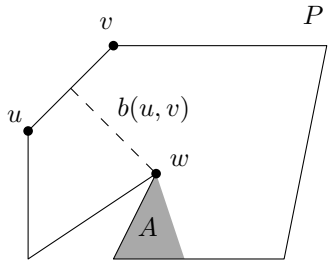


Figure 4: Polygon P is not in general position since $b(u, v)$ intersects vertex w of P. In this case, all points in region A are equidistant from both u and v.

Let $\alpha = \theta/2$ be the angle between the tangent to the bisector at p and $\overline{pu'}$. $\alpha \in [0, \pi]$ since, if this were not the case, then the shortest path from u to p would properly intersect $b(u, v)$. Consider point $p' \in b(u, v)$ infinitesimally close to p in the direction such that p' is clockwise from $\overline{pv'}$ to $\overline{pu'}$. We consider the segment of $b(u, v)$ between p and p' to be a straight line. Let β be the angle between the tangent of $b(u, v)$ at p' and $\overline{p'u'}$. We now have a triangle composed of segments and angles $\overline{pu'}, \alpha, \overline{pp'}, \beta, \overline{p'u'}, \varepsilon$ as shown in Fig. 5. This implies:

$$d(u', p) = \frac{\sin\alpha}{\sin\beta} d(u', p') \qquad \text{Law of Sines}$$

$$\beta = \pi - \alpha - \varepsilon \qquad \text{sum of angles}$$

$$\implies d(u', p) = \frac{\sin\alpha}{\sin(\alpha + \varepsilon)} d(u', p')$$

This implies $d(u', p) > d(u', p')$ for $\alpha < \pi/2$ and $d(u', p) < d(u', p')$ for $\alpha \geq \pi/2$ which implies the same for $d(u, p)$ and $d(u, p')$. Since alpha strictly increases as we move along $b(u, p)$ in the direction of p' from p (increases by ε at each step), $d(u, p)$ is strictly decreasing for $\alpha < \pi/2$ and strictly increasing for $\alpha \geq \pi/2$. □

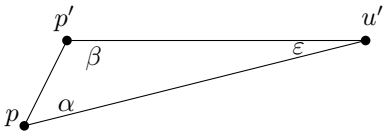


Figure 5: The triangle created by u', p and p' as in Lemma 1.

C Proof of Lemma 3

Lemma 3 *Given a burn region R associated with ignition point u, there exists $p \in R$ such that $d(u, p) = \sup_{p' \in R} d(u, p')$. Moreover, for all such $p \in R$, at least one of the following is true:*

1. *$p \in (V \cap R) \setminus \{u\}$*
2. *p is the intersection of $b(u, v)$ with ∂P for some $v \in V$*

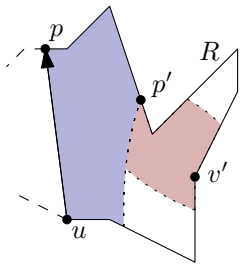


Figure 6: An enclosed region $R = \mathcal{R}(\{u\}, p, (p, u))$ as defined in Lemma 11

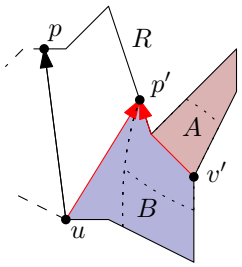


Figure 7: The enclosed regions implied by the existence of v' and p' in Obs. 3. $A = \mathcal{R}(\{v'\}, p', (p', v'))$, $B = \mathcal{R}(\{u, v'\}, p', (v', u))$.

3. p is the intersection of $b(u, v)$ with $b(u, w)$ for some $v, w \in V$

Proof. Fix u and its corresponding burn region R. Existence of p follows directly from 1) the function $d(u, p) : R \longrightarrow \mathbb{R}$ is continuous and 2) R is closed and bounded in $\mathbb{R}^2$ and hence compact. Let $p \in R$ and suppose none of the above is true.

Case 1: p is an interior point of R. In this case, a point $q \in R$ with $d(u, q) > d(u, p)$ can be found by extending the last segment in $\pi(u, p)$ by an infinitesimally small amount and taking the new endpoint as q.

Case 2: p is not an interior point of R. Then $p \in b(u, v)$ for some $v \in V$ or $p \in \partial P$. If p is on some bisector then Lemma 2 implies the existence of $q \in R$ infinitesimally close to p on the same bisector as p such that $d(u, q) > d(u, p)$.

Suppose $p \in \partial P$ and the anchor vertex p' doesn't change at $\pi(u, p)$. We must have $|p'q| > |p'p|$ for all q lying on at least one side of p on the same polygon edge. Then we can find $q \in R$ on the polygon edge infinitesimally close to p such that $|p'q| > |p'p|$ while maintaining p' as the anchor vertex along $\pi(u, q)$. Then $d(u, q) = d(u, p') + |p'q| > d(u, p') + |p'p| = d(u, p)$ and $d(u, p) \neq \sup_{p' \in R} d(u, p')$.

Alternatively, if $p \in \partial P$ and the anchor vertex does change at $\pi(u, p)$. Let p' denote the anchor vertex and p'' denote the second to last vertex on $\pi(u, p)$. Similarly, we must have $|p''q| > |p''p|$ for all q lying on at least one side of p on the same polygon edge. We can find $q \in R$ infinitesimally close to p such that $|p''q| > |p''p|$ and the anchor vertex along $\pi(u, q)$ is either p' or p''. If the anchor vertex is p'' then by argument similar to the previous case, we are done. Otherwise, applying the triangle inequality under Euclidean distance, we have $d(u, q) = d(u, p'') + |p''p'| + |p'q| \geq d(u, p'') + |p''q| > d(u, p'') + |p''p| = d(u, p)$.

$d(u, p) \neq \sup_{p' \in R} d(u, p')$ in all cases if p satisfies none of the above statements. Combined with the fact that some $p \in R$ must satisfy $d(u, p) = \sup_{p' \in R} d(u, p')$, the lemma is true. □

D Algorithm Correctness

Lemma 11 *(see Fig. 6) Given an enclosed region $R = \mathcal{R}(\{u\}, p, (p, u))$, if R does not lie entirely in $\texttt{Vor}(U)[u]$, there exists an ignition point $v' \in \partial P(p, u)$ such that the following properties hold:*

- $\texttt{Vor}(U)[v'] \subset R$
- $p' \in \partial P(p, v') \cap b(u, v')$
- $\partial P(p, p') \subset \texttt{Vor}(U)[u]$

The symmetric claim holds for $R = \mathcal{R}(\{u\}, p, (u, p))$.

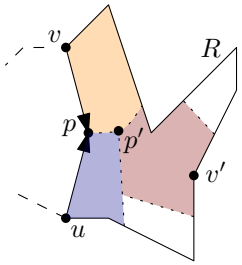


Figure 8: An enclosed region $R = \mathcal{R}(\{u, v\}, p, (v, u))$ that falls into Case 1 of Lemma 12.

Proof. Since R does not lie entirely in $\texttt{Vor}(U)[u]$, there exists a set S of one or more burn regions in R associated with ignition points in $U \setminus \{u\}$. Furthermore, by Lemma 5, since $\pi(u, p)$ separates P, the ignition points must all exist in $\partial P(p, u)$. By Obs. 1, there exist points along $\partial P(p, u)$ that are in $\bigcup S$. Since $p \in \texttt{Vor}(U)[u]$, as we travel along $\partial P(p, u)$ from u to p, we must eventually get to a point p' where $p' \in \bigcup S$ and, for all points $b \in \partial P(p, p')$, $b \in \texttt{Vor}(U)[u]$. We choose v' to be the ignition point associated with a region in S that contains p'. Because $\texttt{Vor}(U)[v']$ does not contain u, it is a proper subset of R. □

Observation 3 *(see Fig. 7) Given the enclosed region $R = \mathcal{R}(\{u\}, p, (p, u))$, the existence of p' and paths $\pi(u, p')$ and $\pi(v', p')$ in R implies the existence of two new enclosed regions derived from $\texttt{Vor}(U)$:*

- $A = \mathcal{R}(\{v'\}, p', (p', v'))$
- $B = \mathcal{R}(\{u, v'\}, p', (v', u))$

The symmetric claim holds for $R = \mathcal{R}(\{u\}, p, (u, p))$.

Lemma 12 *Given an enclosed region $R = \mathcal{R}(\{u, v\}, p, (a, b))$, if R does not lie entirely in $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$, there are two possible cases:*

- *Case 1 (Fig. 8): If $b(u, v)$ intersects some region other than $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$ in R, there exists an ignition point $v' \in U$ such that the following properties hold:*

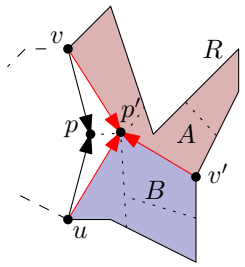


Figure 9: The enclosed regions implied by the existence of v' and p' in Case 1 of Obs. 4. $A = \mathcal{R}(\{v, v'\}, p', (v, v'))$, $B = \mathcal{R}(\{u, v'\}, p', (v', u))$.

  - $\texttt{Vor}(U)[v'] \subset R$
  - $\texttt{Vor}(U)[v']$ *intersects* $\texttt{Vor}(U)[u] \cap \texttt{Vor}(U)[v]$ *at some point* $p' \in R$

- *Case 2:* $b(u, v)$ *does not intersect any region in* R *other than* $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$. *The following properties hold:*
  - *There exists burn paths* $\pi(u, p')$ *and* $\pi(v, p')$ *where* p' *is the point at the intersection of* $b(u, v)$ *with* ∂P *in* R

Proof. Case 1: If $b(u, v)$ intersects regions other than $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$, there exists a region $\texttt{Vor}(U)[v']$ at the point p' on $b(u, v)$ at the boundary of $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$. This point, by definition, is the Voronoi vertex associated with vertices u, v, v' and therefore is in all three associated regions. Since $\pi((, u), p) \cup \pi(v, p)$ separates R from the rest of P, $\texttt{Vor}(U)[v'] \subset R$.

Case 2: $b(u, v)$ intersects $\partial P \cap R$ at some point p' since, if this were not the case, we could draw a path from u to v through R that does not intersect $b(u, v)$. Since $b(u, v)$ does not intersect any region in R other than $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$, $p' \in \texttt{Vor}(U)[u] \cup \texttt{Vor}(U)[v]$. This implies there exists burn paths $\pi(u, p')$ and $\pi(v, p')$ in R. □

Observation 4 *Without loss of generality, assume* R *is enclosed by* $\partial P(v, u)$.

In Case 1 of Lemma 12, the existence of p' *and burn paths* $\pi(u, p')$, $\pi(v, p')$, *and* $\pi(v', p')$ *in* R *implies the existence of two new enclosed regions derived from* $\texttt{Vor}(U)$ *(Fig. 9):*

- $A = \mathcal{R}(\{v, v'\}, p', (v, v'))$
- $B = \mathcal{R}(\{u, v'\}, p', (v', u))$

In case 2 of Lemma 12, the existence of p' *and burn paths* $\pi(u, p')$ *and* $\pi(v, p')$ *implies the existence of two new enclosed regions derived from* $\texttt{Vor}(U)$

- $A = \mathcal{R}(\{v\}, p', (v, p'))$
- $B = \mathcal{R}(\{u\}, p', (p', u))$

Observation 5 *The region* $R_{u,v} = R \setminus (A \cup B)$ *is enclosed by burn paths* $\pi(u, p')$, $\pi(u, p)$, $\pi(v, p')$, *and* $\pi(v, p)$. *Furthermore,* $R_{u,v}$ *lies entirely in* $\texttt{Vor}(U)[u] \cup \texttt{Vor}(U)$ *and is isolated from* $A \cup B$ *by* $\pi(u, p') \cup \pi(v, p')$.

Theorem 10 *Given an enclosed region* $R = \mathcal{R}(N, p, (a, b))$, *a time* $t(p_0)$ *and* v' *as the next enclosing vertex, if it is possible, the Next Enclosing Vertex Algorithm gives us two enclosed regions* A *and* B *separated by* $\pi(v', p')$, *such that* A *and* B *being properly burned in time* $t(p_0)$ *by a set of ignition points* U *is equivalent to* R *being properly burned in time* $t(p_0)$ *and paths* $\pi(N, p')$ *and* $\pi(v', p')$ *being burn paths in* $\texttt{Vor}(U)$. *If it is not possible the Next Enclosing Vertex Algorithm returns nothing.*

Proof. In both cases, A and B are separated by $\pi(v', p')$ (Obs. 3 & 4). This implies that if A and B are properly burned, v' must be an ignition point.

If $d(v', p) < d(N, p)$, p would not be in the Voronoi regions associated with enclosing vertices in N which means the enclosing vertices and enclosing point of R would not be preserved if v' is chosen as an ignition point (Step 1). In addition, if v' implies an enclosing point p' that is not burned by v' in time $t(p_0)$, v' cannot be the next enclosing vertex (Step 2a for both cases).

- Case 1: Without loss of generality, assume $R = \mathcal{R}(\{u\}, p, (p, u))$.
  $p' \in \partial P(p, v)$ since if this was not the case, it would imply $d(v', p) < d(u, p)$ since $b(u, v')$ would properly intersect $\pi(u, p)$ which would imply p is closer to v' than u. If A and B are properly burned in time $t(p_0)$, it must be the case that the burn path $\pi(u, p')$ exists. Let R_u be the region enclosed by burn paths $\pi(u, p')$ and $\pi(u, p)$ and $\partial P(p, u)$. By Obs. 1 (star-shaped property), R_u lies entirely in $\texttt{Vor}(U)[u]$. This means $\pi(u, p')$ separates $\pi(u, p)$ from $A \cup B$ which implies that $\pi(u, p)$ is preserved. $\partial P(p, u)$ must be burned by u in time $t(p_0)$ (Step 2b). Since burn paths $\pi(u, p')$ and $\pi(u, p)$ exist and $\partial P(p, p')$ can be burned by u in time $t(p_0)$, by Lemma 1, R_u is entirely burned by u in time $t(p_0)$. This means $R = A \cup B \cup R_u$ is entirely burned in time $t(p_0)$ which implies R is properly burned.
- Case 2: Without loss of generality, assume $R = \mathcal{R}(\{u, v\}, p, (v, u))$.
  $p' \in R$ since if this was not the case, similar to the previous argument, v' would be too close to p. If A and B are properly burned in time $t(p_0)$, it must be the case that burn paths $\pi(u, p')$ and $\pi(v, p')$ exist. This implies that burn paths $\pi(u, p)$ and $\pi(v, p)$ are preserved since they are isolated from A and B by $\pi(u, p') \cup \pi(v, p')$ (Obs. 5. Since p and p' are burned by u and v in time $t(p_0)$, the segment S of $b(u, v)$ between p and p' is burned by u and v by Lemma 1. Let $R_{u,v}$ be the region enclosed by burn paths $\pi(u, p')$, $\pi(u, p)$, $\pi(v, p')$, and $\pi(v, p)$. By Obs. 5, $R \setminus (A \cup B) \subset R_{u,v}$. Similar to Case 1, since S is burned by u and v, the region enclosed by burn paths $\pi(u, p)$ and $\pi(u, p')$ and S is entirely burned by u (and similar for v) (Lemma 1). This implies $R_{u,v}$ is entirely burned by u and v in time $t(p_0)$. This means $R = A \cup B \cup R_{u,v}$ is entirely burned in time $t(p_0)$ which implies R is properly burned.

In both cases, if either A or B cannot be properly burned in time $t(p_0)$, since each region is isolated by known burn paths, R cannot be entirely burned in time $t(p_0)$ with v' as the next enclosing vertex. □

E Determining if an enclosed region is entirely burned by a set of enclosing vertices in time $t(p_0)$

This method takes the arguments to an enclosed region $R = \mathcal{R}(\{N\}, p, (a, b))$ and a time t to burn the enclosed region with N. We assume that p is burned by N in time t.

Implementation:

There are two cases depending on $|N|$:

- $|N| = 1$**:** If $d(u, v') \leq t$ for all $v' \in (V \cap \partial P(a, b)) \cup \{p\}$, return $true$; else return $false$.
- $|N| = 2$**:** It must be the case that $N = \{a, b\}$. Assume, without loss of generality that $u = a$ and $v = b$.

  Let p' be the Type II potential final burn point associated with u and v (clockwise from u to v).

  If $d(u, u') \leq t$ for all $u' \in (V \cap \partial P(u, p')) \cup \{p, p'\}$ and $d(v, v') \leq t$ for all $v' \in V \cap \partial P(p', v)$, return $true$; else return $false$

In both cases, we check all potential points p as in Lemma 3. If all of these points are within t of N, we know that R is entirely covered or we would violate the lemma. If one of the points isn't covered, then R is not covered by N.

Maximum Overlap Area of Several Convex Polygons Under Translations

Hyuk Jun Kweon* Honglin Zhu†

Abstract

Let $k \geq 2$ be a constant. Given any k convex polygons in the plane with a total of n vertices, we present an $O(n \log^{2k-3} n)$-time algorithm that finds a translation of each of the polygons such that the area of intersection of the k polygons is maximized. Given one such placement, we also give an $O(n)$-time algorithm which computes the set of all translations of the polygons which achieve this maximum.

1 Introduction

Shape matching is a critical area in computational geometry, with overlap area or volume often used to measure the similarity between shapes when translated. In this paper, we present a quasilinear time algorithm to solve the problem of maximizing the overlap area of several convex polygons, as stated in the following theorem.

Theorem 1 *Let $P_0, P_1, \ldots, P_{k-1}$ be convex polygons, with a total of n vertices, where k is a constant. Then in $O(n \log^{2k-3} n)$-time, we can find a placement $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{k-1})$ maximizing the area of*

$$(P_0 + \mathbf{v}_0) \cap \cdots \cap (P_{k-1} + \mathbf{v}_{k-1}).$$

Once we have found a placement $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{k-1})$ that maximizes the overlap area, we can compute the set of all such placements in linear time.

Theorem 2 *With the notation in Theorem 1, suppose that we have found a placement $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{k-1})$ maximizing the overlap area. Then in $O(n)$-time, we can compute the set of all placements that maximize the overlap area. This set is represented in terms of $O(n)$ linear constraints without redundancy.*

It is important to note that a greedy method may not yield the optimal result. To illustrate this, consider the following polygons.

1. An equilateral triangle with height 1
2. An upside-down equilateral triangle with height 1
3. A rectangle with height 1 and sufficiently small width ε

If we first place the two triangles to maximize their intersection, and then place the rectangle to maximize the overlap with the combined shape, the resulting overlap area is approximately $(2/3)\varepsilon$. However, the true maximum overlap area of all three shapes is approximately ε. This example highlights the necessity of our algorithm.

Suppose that we have k polytopes in $\mathbb{R}^d$ with n vertices in total. Clearly, the overlap volume function under translation is a piecewise polynomial function. To find the maximum overlap volume under translation, we can compute the maximum on each piece. For example, Fukuda and Uno presented an $O(n^4)$-time algorithm for maximizing the overlap area of two polygons in $\mathbb{R}^2$ [9, Theorem 6.2]. They also gave an $O((kn^{dk+1})^d)$-time algorithm for the problem with k possibly non-convex polytopes in $\mathbb{R}^d$ [9, Theorem 6.4].

If the polytopes are convex, then the overlap volume function is log-concave. This follows immediately from two properties: the 0-1 indicator functions of convex sets are log-concave, and the product of two log-concave functions is again log-concave. With this additional structure, one may apply a prune-and-search technique and make the algorithm much faster. For example, de Berg et al. gave an $O(n \log n)$-time algorithm to find the maximum overlap of two convex polygons in $\mathbb{R}^2$, which is highly practical due to the small constant hidden in the order notation [7, Theorem 3.8]. Ahn, Brass and Shin gave a randomized algorithm for finding maximum overlap of two convex polyhedrons in expected time $O(n^3 \log^4 n)$ [1, Theorem 1]. Ahn, Cheng and Reinbacher [2, Theorem 2] find an $O(n \log^{3.5} n)$-time algorithm for the same problem after taking a generic infinitesimal perturbation. The last two results cited from [1] and [2] have also been generalized to higher-dimensional cases within the same papers.

On the other hand, there are few known results for problems involving several convex shapes. In this regard, Zhu and Kweon proposed an $O(n \log^3 n)$-time algorithm to find the maximal overlap area of three convex polygons [16, Theorem 1.2]. This result is based on an $O(n \log^2 n)$-time algorithm that finds the maximum overlap area of a convex polyhedron and a convex polygon in $\mathbb{R}^3$ [16, Theorem 1.1]. The main algorithm of this paper is a strict generalization of both [7, Theorem 3.8] and [16, Theorem 1.2].

*Department of Mathematics, University of Georgia, kweon@uga.edu, This author is funded by The AMS-Simons Travel Grant program.

†Department of Mathematics, Massachusetts Institute of Technology, honglinz@mit.edu

2 Notation and Terminology

In this paper, we use the notation $\operatorname{Supp} f$ to refer to the closed support of a function f, i.e., the closure of the set of points where f is nonzero. For two sets $A, B \in \mathbb{R}^d$, we define their Minkowski sum and difference as $A + B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$ and $A - B = \{\mathbf{x} \mid \mathbf{x} + B \subset A\}$, respectively.

We consider closed polytopes unless otherwise specified. When referring to a polytope P, its (geometric) interior consists of the set of points not on the facets, while its (geometric) boundary comprises the set of points on the facets. On the other hand, the topological interior of $P \subset R^n$ is the set of points in P that have an open ball entirely contained in P. The topological boundary of P consists of points that are on the interior of P. For example, if P is a polygon in $\mathbb{R}^3$, then its geometric boundary consists of the edges, which are one-dimensional, while its topological boundary corresponds to the polygon itself.

The computation model is based on the real RAM model, yet the base field $\mathbb{R}$ can be substituted with any ordered field R. Specifically, we assume that within R, the binary operations $+$, $-$, $\times$, and $/$, as well as the binary relations $<$ and $=$, can be precisely computed in constant time.

This generalization is important since we employ the technique of symbolic infinitesimal translation, similar to [7]. Whenever a new infinitesimal number $\varepsilon > 0$ is introduced, we change our base field from R to the functor field $R(\varepsilon)$. One may have concerns about this, since operations in $R(\varepsilon)$ cannot be generally performed in constant time. However, as our algorithm only computes quadratic functions, we use ε^n only for $n \leq 2$. Furthermore, we introduce fewer than $2k$ infinitesimal numbers. Therefore, we may assume that operations in $R(\varepsilon)$ can be executed in constant time. The assumption regarding the constant-time operations in $R(\varepsilon)$ is technical and not significant in the context of the conceptual framework. Therefore, we will not delve into a meticulous explanation of this aspect.

3 Configuration Space

The aim of this section is to define the configuration space, the domain of the overlap area function, and discuss its properties. Throughout the paper, we take k convex polygons $P_0, P_1, \ldots, P_{k-1}$, where k is a constant. Let $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1} \in \mathbb{R}^2$ be vectors of indeterminates. The overlap area of

$$I = (P_0 + \mathbf{v}_0) \cap (P_1 + \mathbf{v}_1) \cap \cdots \cap (P_{k-1} + \mathbf{v}_{k-1})$$

is invariant under the map

$$(\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}) \mapsto (\mathbf{v}_0 + \mathbf{x}, \ldots, \mathbf{v}_{k-1} + \mathbf{x}).$$

Therefore, we define the configuration space as a $(2k-2)$-dimensional quotient linear space

$$\mathcal{C} := \frac{\{(\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}) \colon \mathbf{v}_i \in \mathbb{R}^2\}}{\{(\mathbf{x}, \ldots, \mathbf{x}) \colon \mathbf{x} \in \mathbb{R}^2\}}.$$

One may also define the configuration space by fixing one polygon, but this definition loses symmetry and makes the algorithm more complicated. Any element of $\mathcal{C}$ will be called a placement. We denote $(\mathbf{v}_0; \ldots; \mathbf{v}_{k-1}) \in \mathcal{C}$ as a placement that corresponds to $(\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}) \in (\mathbb{R}^2)^k$.

We define the overlap area function $\Pi \colon \mathcal{C} \to [0, \infty)$ as

$$\Pi(\mathbf{v}_0; \ldots; \mathbf{v}_{k-1}) := |(P_0 + \mathbf{v}_0) \cap \cdots \cap (P_{k-1} + \mathbf{v}_{k-1})|.$$

and then its support $\operatorname{Supp} \Pi$ is compact. To compute $\Pi(\mathbf{v}_0; \ldots; \mathbf{v}_{k-1})$ in linear time, we use the following theorem:

Theorem 3 (Shamos) *Let P and Q be convex polygons of m and n vertices, respectively. Then $P \cap Q$ can be computed in $O(m+n)$-time.*

Proof. This was first proved by Shamos [15, Section 5.2]; see also [14, Section 7.6]. □

The vertices $(x_0, y_0), \ldots, (x_{r-1}, y_{r-1})$ of the overlap I can be expressed as linear functions in $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}$ in a generic setting. Ordering them in counter-clockwise direction, the area of I can be computed using the shoelace formula:

$$|I| = \frac{1}{2} \sum_{i \in \mathbb{Z}/r\mathbb{Z}} (x_i y_{i+1} - x_{i+1} y_i),$$

where the indices are taken modulo m. Therefore, Π is a piecewise quadratic function of $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}$.

Note that Π may not be quadratic in two cases:

(I) an edge of a polygon $P_i + \mathbf{v}_i$ contains a vertex of another polygon $P_j + \mathbf{v}_j$ and

(II) edges of three distinct polygons $P_i + \mathbf{v}_i$, $P_j + \mathbf{v}_j$ and $P_k + \mathbf{v}_k$ intersect at one point.

Each of these events defines a polytope in $\mathcal{C}$ of codimension 1. Following [7], we call such a polytope as an event polytope. An event polytope defined by (I) (resp. (II)) is called of type I (resp. of type II). A hyperplane containing a type I (resp. type II) event polytope is also called of type I (resp. of type II). There are $O(n^2)$ type I hyperplanes and $O(n^3)$ type II hyperplanes.

4 Linear Programming

Let $L \subset \mathcal{C}$ be an r-flat. The goal of this section is to provide an $O(n)$-time algorithm that finds a placement $\mathbf{v} \in L$ such that

$$\Pi(\mathbf{v}) \neq 0.$$

If no such placement exists, the algorithm returns **None**.

When working with two polygons, $\operatorname{Supp}\Pi$ is simply the Minkowski sum $P_0 + (-P_1)$, where $-P_1$ is the polygon P_1 reflected about the origin. However, when working with more than two polygons, the problem becomes more complex. To tackle this problem, we use linear programming with Meggido's solver.

Theorem 4 (Megiddo [11]) *If the number of variables is fixed, a linear programming problem with n constraints can be solved in $O(n)$-time.*

Let n_i be the number of vertices of P_i. Then P_i is defined by n_i linear inequalities:

$$f_{i,a}(\mathbf{x}) \geq 0 \quad (\text{for } a < n_i).$$

The codimension of the r-flat $L \subset \mathcal{C}$ is $2k-r-2$. Thus, L is defined by $2k-r-2$ linear equations:

$$g_b(\mathbf{v}) = 0 \quad (\text{for } b < 2k-r-2).$$

Then a placement $\mathbf{v} = (\mathbf{v}_0; \dots; \mathbf{v}_{k-1}) \in \mathcal{C}$ a point $\mathbf{x} \in \mathbb{R}^2$ and satisfy the constraints

$$\begin{cases} f_{i,a}(\mathbf{x} - \mathbf{v}_i) \geq 0 & (\text{for } i < k \text{ and } a < n_i) \text{ and} \\ g_b(\mathbf{v}) = 0 & (\text{for } b < 2k-r-2). \end{cases} \tag{1}$$

if and only if $\mathbf{x} \in (P_0 + \mathbf{v}_0) \cap \dots \cap (P_{k-1} + \mathbf{v}_{k-1})$ and $\mathbf{v} \in L$. Therefore, we obtain the lemma below.

Lemma 5 *We have $\mathbf{v} \in L \cap \operatorname{Supp}\Pi$ if and only if $(\mathbf{x}, \mathbf{v})$ satisfies (1) for some point $\mathbf{x}$ in a plane.*

Hence, in $O(n)$-time, we can get $\mathbf{v} \in L \cap \operatorname{Supp}\Pi$, by solving any linear programming with the constraints (1). One problem is that $\mathbf{v}$ might be on the (topological) boundary of $\operatorname{Supp}\Pi$.

Lemma 6 *Let M be the solution set of linear constraints*

$$\begin{cases} p_i(\mathbf{x}) \geq 0 & (\text{for } i < n) \text{ and} \\ q_j(\mathbf{x}) = 0 & (\text{for } j < m) \end{cases} \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^d$ and d is constant. Then we can compute the maximal affinely independent finite subset S of M in $O(m+n)$-time.

Proof. The proof can be found in the appendix. □

Theorem 7 *In $O(n)$-time, we can either return $\mathbf{v} \in L$ such that $\Pi(\mathbf{v}) \neq 0$, or return* **None** *if none exists.*

Proof. Let $M \subset \mathbb{R}^2 \times L$ be the solution set of the constrains (1). Then $\Pi(\mathbf{v}) \neq 0$, if and only if $(\mathbf{x}, \mathbf{v})$ is an topological interior point of $M \subset \mathbb{R}^2 \times L$ for some $\mathbf{x} \in \mathbb{R}^2$. Applying Lemma 6, we get the maximal affinely independent set S of M.

If $|S| \leq r+2$, then $\dim M < 2 + \dim L$, and M has no topological interior point, so we return **None**. If $|S| = r+3$, then

$$(\mathbf{x}_{\text{avg}}, \mathbf{v}_{\text{avg}}) = \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{v}) \in S} (\mathbf{x}, \mathbf{v})$$

is an topological interior points of $M \subset \mathbb{R}^2 \times L$. Hence, we return $\mathbf{v}_{\text{avg}}$. □

5 Decision Problem

We aim to find the maximum of Π on an r-flat $L \subset \mathcal{C}$ using an induction on r. To do so, we apply a prune-and-search technique on the set of event polytopes. However, this technique requires solving a decision problem: given a hyperplane $H \subset L$, we must determine on which side of H the maximum of $\Pi|_L$ lies. In this section, we provide an algorithm for this decision problem under certain induction hypotheses.

Theorem 8 *The square root of $\Pi\colon \mathcal{C} \to [0, \infty)$ is concave on its support.*

Proof. This follows from the Brunn–Minkowski inequality [13][4]; see also [9, Theorem 3.3]. □

Now, we assume the following hypothesis for some non-negative function $T(n)$ in the rest of this section.

Hypothesis 9 *Let $L \subset \mathcal{C}$ be an $(r-1)$-flat. Then we can find $\mathbf{v} \in L$ maximizing $\Pi|_L$ in $O(T(n))$-time.*

We can partition L into open polytopes on which Π is quadratic. Therefore, the maximum $\mathbf{v} \in L$ of $\Pi|_L$ is a placement.

Theorem 10 *Given an r-flat L and its hyperplane $H \subset L$, let $M \subset L$ be the set of maximum points of $\Pi|_L$. We can determine which side of H contains M in $O(T(n))$-time.*

Proof. Let ε be an positive infinitesimal smaller than any positive number in the base field in R. For any $t \in R(\varepsilon)$, let

$$h(t) = \max_{\mathbf{v} \in t\mathbf{n}+H} \Pi(\mathbf{v}).$$

Let $N \subset R(\varepsilon)$ be the set of all maximum points of $h(x)$. It suffices to decide on which side N lies with respect to 0. By Theorem 8, the function $h\colon R(\varepsilon) \to [0, \infty)$ is unimodal.

By Hypothesis 9, we can compute the sequence

$$S = \big(h(-\varepsilon_{s+1}), h(0), h(\varepsilon_{s+1})\big)$$

in $O(T(n))$-time. If $h(0) = 0$, then all interior points of $\operatorname{Supp} h$ lie in the same side with respect to 0. In this case, apply Lemma 7 and attempt to get one point of $\operatorname{Supp} h$. If $h(0) \neq 0$, there are three remaining cases.

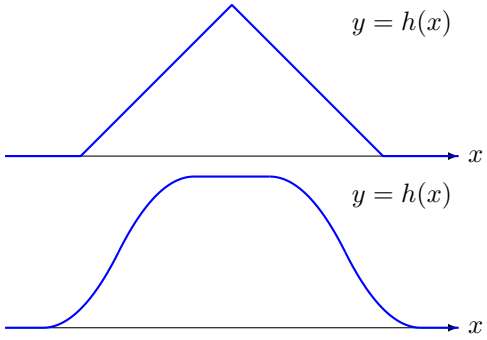


Figure 1: Two possible examples of the graph of h

1. If S is strictly increasing, then $N \subset (0, \infty)$.
2. If S is strictly decreasing, then $N \subset (-\infty, 0)$.
3. If S is not strictly monotonic, then $0 \in N$.

This completes the proof. □

This proof highlights the necessity of infinitesimal translations for our algorithm. Given that infinitesimal numbers are introduced only in this step, we introduce no more than $\dim \mathcal{C} = 2k - 2$ infinitesimal numbers.

6 Two Polygons

The goal of this section is to present a linearithmic time algorithm for finding a translation that maximizes the overlap area of two convex polygons under translations. This problem was previously studied by de Berg et al. [7, Theorem 3.8], but our approach is different and allows for handling multiple polygons.

In this section, we only have two convex polygons $P = P_0$ and $Q = P_1$ with n and m vertices, respectively. We consider only one translation vector $\mathbf{v} = \mathbf{v}_1 - \mathbf{v}_0$, and since $\mathcal{C}$ is two-dimensional, we refer to event polytopes and hyperplanes as event line segments and lines, respectively. Since there are no type II line segments, all event line segments can be defined by one of the following two events:

1. an edge of a polygon P contains a vertex of polygon $Q + \mathbf{v}$ and
2. an edge of a polygon $Q + \mathbf{v}$ contains a vertex of polygon P.

The first type of event lines segment will be called of type $(0, 1)$ and the second type of event lines will be called type $(1, 0)$ line segments. The same rules apply to event lines.

Type $(0, 1)$ lines are organized into n groups, each with m parallel lines. Our goal is to efficiently prune this set, requiring an appropriate representation. We use 'arrays' to denote sequential data structures with constant time random access, and assume the size of each array is predetermined.

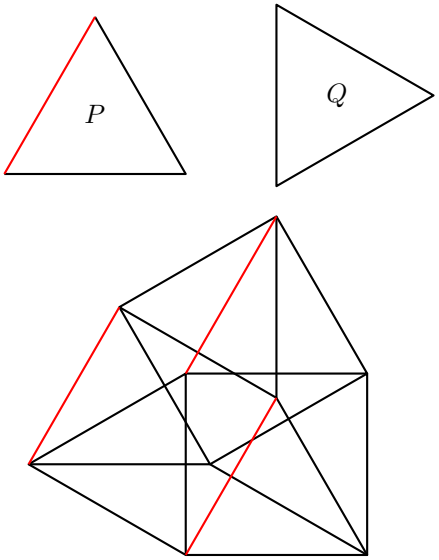


Figure 2: Event line segments. The parallel lines of one group are highlighted in red.

The n groups of parallel lines are represented by sorted arrays $A_0, A_1, \ldots, A_{n-1}$. Each array A_i holds the y-intercepts and a single slope value for the lines in the i-th group. For vertical lines in A_i, we store the x-intercepts instead.

Definition 11 *A slope-intercept array A consists of sorted arrays $A_0, A_1, \ldots, A_{n-1}$, with each A_i associated with a slope that is a potentially infinite number. Its number of groups is n, and its size $|A|$ is the sum of the sizes of A_i. Another slope-intercept array A' is a pruned array of A if it consists of A with identical slopes.*

We can use [16, Theorem 1.4] to prune a slope-intercept array A, but the description is complicated and the result is weaker. Instead, we rely on a stronger version, which we prove in the appendix.

Theorem 12 *For a slope-intercept array A with n groups of lines, we can partition the plane $\mathbb{R}^2$ into four closed quadrants $T_0, \ldots, T_3$ using one horizontal line ℓ_0 and one non-horizontal line ℓ_1. Additionally, for each $i < 4$, we can compute pruned array P_i of A that include all lines intersecting the interior of P_i and have size at least $(3/4)|A|$, all in $O(n)$-time.*

Now, we will represent the set of type $(0, 1)$ event lines using a slope-intercept array.

Lemma 13 *We have n linear functions $f_0, \ldots, f_{n-1}$ and m vertices $v_0, \ldots, v_{m-1}$ of a convex polygon, both ordered counterclockwise by their gradient vectors and arrangement, respectively. In $O(m + n)$-time, we can*

find indices $a(0), \dots, a(n-1)$ such that vertex $v_{a(i)}$ minimizes $f_i(v_j)$ for all $j < m$.

Proof. In $O(m)$-time, we can find $a(0)$ by computing all $f_0(v_j)$. Now, suppose that $a(i-1)$ is computed. Then compute the sequence

$$f_i(v_{a(i-1)}), f_i(v_{a(i-1)+1}), f_i(v_{a(i-1)+2}), \dots$$

until it increases after some index a'. Then $f_i(v_{a'})$ maximizes f_i, so $a(i) = a'$. By repeating this process, we can find all $a(0), a(1), \dots, a(m-1)$. Observe that $v_{a(0)}, v_{a(1)}, \dots, v_{a(n-1)}$ are sorted counterclockwise. Since we only perform one rotation, this process requires $O(m+n)$-time. □

Lemma 14 *In $O(m+n)$-time, we can construct a slope-intercept array of $2n$ groups of size mn representing the set of all type $(0,1)$ lines*

Proof. The proof can be found in the appendix. □

Theorem 15 *Let P and Q be convex polygons, with m and n vertices, respectively. In $O((m+n)\log(m+n))$-time, we can finds a translation $\mathbf{v} \in \mathbb{R}^2$ maximizing the overlap area*

$$\Pi(\mathbf{v}) = |P \cap (Q + \mathbf{v})|.$$

Proof. For any line $\ell \subset \mathbb{R}^2$, we can compute a point $\mathbf{v} \in \ell$ maximizing $\Pi|_\ell$ in $O(m+n)$-time by [3, Corollary 4.1]. Using Theorem 10, we can determine on which side of ℓ the set of maxima of Π lies in $O(m+n)$-time.

By constructing a slope-intercept array A of $(m+n)$ groups with Lemma 14, we can represent all event lines in $O(m+n)$-time. Applying Theorem 10 to ℓ_0 and ℓ_1 obtained from Theorem 12, we can prune A to about $1/8$ of its size, and this step requires $O(m+n)$-time. After $O(\log(m+n))$ steps, only $O(1)$ lines remain, and we can find a placement $\mathbf{v}$ that maximizes the overlap area $\Pi(\mathbf{v})$ directly. □

7 Several Polygons

The aim of the section is to give an $O(n \log^{2k-3} n)$-time algorithm to compute $\mathbf{v} \in \mathcal{C}$ maximizing Π. We first restrict the domain of Π into an r-flat $L \subset \mathcal{C}$ and prove a slightly stronger statement below by induction on r.

Theorem 16 *Let $L \subset \mathcal{C}$ be an r-flat. Then we can find $\mathbf{v} \in L$ maximizing $\Pi|_L$ in $O(n \log^{r-1} n)$-time.*

The proof of the base case can be obtained by modifying the proof of [3, Corollary 4.1].

Lemma 17 *Let $\ell \subset \mathcal{C}$ be a line. Then in $O(n)$-time, we can find $\mathbf{v} \in \ell$ maximizing $\Pi|_\ell$.*

Proof. The proof can be found in the appendix. □

Therefore, we assume that $r > 1$ and the following induction hypothesis is true.

Hypothesis 18 *Let $L \subset \mathcal{C}$ be an $(r-1)$-flat. Then we can find $\mathbf{v} \in L$ maximizing $\Pi|_L$ in $O(n \log^{r-2} n)$-time.*

We will first find an r-simplex $T_I \subset L$ such that T_I has the maximum point of $\Pi|L$ and no type I hyperplane intersects the interior of T_I. Recall that type I hyperplanes are defined by the following event.

(I) an edge of a polygon $P_i + \mathbf{v}_i$ contains a vertex of another polygon $P_j + \mathbf{v}_j$

If i and j are specified, then it will be called a type (i,j) hyperplane. Then type I hyperplanes are grouped into $k(k-1)$ groups, each of which is the set of type (i,j) hyperplanes. Any type (i,j) hyperplane H is defined by a linear equation of the form

$$\mathbf{n} \cdot (\mathbf{x}_i - \mathbf{x}_j) = c$$

for some $\mathbf{n} \in \mathbb{R}^2$ and $c \in \mathbb{R}$. Consider the projection

$$\begin{aligned} \pi_{i,j} \colon \mathcal{C} &\to \mathbb{R}^2 \\ \mathbf{x} &\mapsto \mathbf{x}_i - \mathbf{x}_j. \end{aligned}$$

Then $\pi_{i,j}(H) \subset \mathbb{R}^2$ is a line. Such a line will also be called of type (i,j). Thus, we will find a triangle $T_{i,j} \subset L$ such that no type (i,j) lines intersect the interior of $T_{i,j}$.

Proposition 19 *In $O(n \log^{r-1} n)$-time, We can find a triangle $T_{i,j} \subset \mathbb{R}^2$ such that*

1. *a maximum point of $\Pi|_L$ lies on $\pi_{i,j}^{-1}(T_{i,j}) \cap L$, and*
2. *no type (i,j) lines intersects the interior of $T_{i,j}$.*

Proof. The proof is similar to that of Theorem 15 and can be found in the appendix. □

Now, define

$$T_I \coloneqq \bigcap_{i,j<d} \pi_{i,j}^{-1}(T_{i,j}) \subset L. \tag{3}$$

Then T_I is defined by $3k(k-1) \in O(1)$ linear functions, and by construction, no type I hyperplanes intersect the interior of T_I. Our goal now is to find an r-simplex $T \subset T_I$ such that T has the maximum point of $\Pi|_L$ and no event polytopes intersect the interior of T.

To achieve this, we first note that only $O(n)$ type II hyperplanes intersect the interior of T_I. Thus, we can obtain T by repeatedly applying Chazelle's cutting algorithm.

Definition 20 (Matoušek [10]) *A cutting of $\mathbb{R}^d$ is a collection C of possibly unbounded d-simplices with disjoint interiors, which together cover $\mathbb{R}^d$. Let S be a set of n hyperplanes in $\mathbb{R}^d$. Then a cutting C is a $(1/2)$-cutting for S if the interior of each simplex intersects at most $n/2$ hyperplanes.*

Theorem 21 (Chazelle [5]) *With the notation provided in Definition 20, where d is constant, a $(1/2)$-cutting of size $O(1)$ can be computed in $O(n)$-time. In addition, the set of hyperplanes intersecting each simplex of the cutting is reported in the same time.*

Proposition 22 *In $O(n \log^{r-1} n)$-time, we can find an r-simplex $T \subset L$ such that*

1. *the maximum point of $\Pi|_L$ lies on T, and*
2. *no event polytope intersects the interior of T.*

Proof. Take T_I as defined in (3). By construction, no type I hyperplane intersects the interior of $T_I \subset L$. Therefore, the set of pairs of intersecting edges of P_i and P_j does not depend on the placement $\mathbf{v} \in T_I$. Moreover, every edge of P_i intersects at most two edges of P_j. Therefore, there are at most

$$\binom{k}{3} 4n \in O(n)$$

type II polytopes intersecting the interior of T_I. In $O(n)$-time, we can compute the set S containing all such type II hyperplanes by sampling a placement $\mathbf{v}$ in the interior of T_I.

To find a simplex T satisfying the conditions of Proposition 22, we first set $T = T_I$. Then we define S as the set of hyperplanes in L containing a facet of T or a type II polytope that intersects the interior of T. We can compute a $(1/2)$-cutting C of size $O(1)$ for S in $O(n)$-time using Theorem 21. Using Theorem 10, we can then find a simplex $T' \in C$ containing the maximum point of $\Pi|_L$ in $O(n \log^{r-2} n)$-time. We set $T = T'$ and repeat this process $O(\log n)$-times until no type II polytopes intersect the interior of T. □

We can now prove Theorem 16.

Proof. We can find T as in Proposition 22 and compute $\Pi|_T$, which is a quadratic polynomial. Then we can directly compute the maximum point of $\Pi|_T$. □

Theorem 1 *Let $P_0, P_1, \ldots, P_{k-1}$ be convex polygons, with a total of n vertices, where k is a constant. Then in $O(n \log^{2k-3} n)$-time, we can find a placement $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{k-1})$ maximizing the area of*

$$(P_0 + \mathbf{v}_0) \cap \cdots \cap (P_{k-1} + \mathbf{v}_{k-1}).$$

Proof. This is a corollary of Theorem 16 with $r = 2k - 2$. □

8 Set of Maxima

Our next step is to determine the set $M \subset \mathcal{C}$ of placements $\mathbf{v} \in \mathcal{C}$ that maximize the overlap area Π. Once we identify at least one such placement, the problem becomes easy, as every maximal overlap is the same up to translation. To accomplish this, we rely on the equality condition of the Brunn-Minkowski inequality.

Theorem 23 (Minkowski) *Let A and B be compact subsets of $\mathbb{R}^2$ with nonzero area. Then*

$$\left|\frac{1}{2}A + \frac{1}{2}B\right|^{1/2} \geq \frac{1}{2}|A|^{1/2} + \frac{1}{2}|B|^{1/2},$$

and the equality holds if and only if A and B are homothetic.

We define $I(\mathbf{v})$ for any placement $\mathbf{v} \in \mathcal{C}$, as follows:

$$I(\mathbf{v}) \coloneqq (P_0 + \mathbf{v}_0) \cap \cdots \cap (P_{k-1} + \mathbf{v}_{k-1}).$$

Lemma 24 *Let $\mathbf{v}, \mathbf{u} \in \mathcal{C}$ be two placements that both maximize Π. Then $I(\mathbf{u})$ and $I(\mathbf{v})$ are equivalent up to translation.*

Proof. The proof can be found in the appendix. □

We then fix a maximal overlap $I_{\max} \subset \mathbb{R}^2$. The set of all $\mathbf{v}_i$ such that $I_{\max} \subset \mathbf{v}_i + P_i$ is given by the Minkowski difference

$$\begin{aligned}(-P_i) - (-I_{\max}) &= \{\mathbf{x} \in \mathbb{R}^2 \mid \mathbf{x} + (-I_{\max}) \subset -P_i\} \\ &= \{\mathbf{x} \in \mathbb{R}^2 \mid I_{\max} \subset \mathbf{x} + P_i\}.\end{aligned}$$

We define $N \coloneqq \prod_{i<m}(P_i - I_{\max})$ and let $\pi \colon (\mathbb{R}^2)^k \to \mathcal{C}$ be the natural quotient.

Lemma 25 *The restricted map $\pi|_N \colon N \to M$ is an affine isomorphism.*

Proof. By construction $M = \pi(N)$. Suppose there exist two distinct $\mathbf{u}, \mathbf{v} \in N$ such that

$$\mathbf{u} = \mathbf{v} + (\mathbf{x}, \mathbf{x}, \ldots, \mathbf{x})$$

for some $\mathbf{x} \in \mathbb{R}^2$. This implies that $I_{\max} = I(\mathbf{v})$ and $I_{\max} = I(\mathbf{u}) = I(\mathbf{v}) + \mathbf{x}$. As a result, we must have $\mathbf{u} = \mathbf{v}$. □

Since each P_i and $I_{\max}$ contain at most n vertices, we can represent $(-P_i) - (-I_{\max})$ using $O(n)$ linear constraints without redundancy. This computation can be completed in $O(n)$-time. Consequently, by employing standard linear algebra techniques, we can describe $M \subset \mathcal{C}$ using $O(n)$ linear constraints without redundancy in $O(n)$-time.

Theorem 26 *In $O(n)$-time, we can represent $M \subset \mathcal{C}$ using $O(n)$ linear constraints without redundancy.*

Proof. Let $\mathbf{v}_i = (x_i, y_i)$ for each $i < m$. A linear function $f(\mathbf{v}_0, \ldots, \mathbf{v}_{k-1})$ can be written as an affine combination of $\mathbf{v}_1 - \mathbf{v}_0, \ldots, \mathbf{v}_{k-1} - \mathbf{v}_0$ if and only if

$$\sum_{i<m} \frac{\partial}{\partial x_i} f = 0 \quad \text{and} \quad \sum_{i<m} \frac{\partial}{\partial y_i} f = 0.$$

Every edge of $I_{\max}$ should be part of an edge of P_i for some $i < m$. Consider two nonparallel edges. They yield two linear equations:

$$\mathbf{a} \cdot \mathbf{v}_i = c \quad \text{and} \quad \mathbf{b} \cdot \mathbf{v}_j - d.$$

Here, $\mathbf{v}_i$ and $\mathbf{v}_j$ are column vectors, and $\mathbf{a}$ and $\mathbf{b}$ are row vectors. Let

$$\mathbf{v}' = \begin{pmatrix} x' \\ y' \end{pmatrix} \coloneqq \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{a} \cdot \mathbf{v}_i - c \\ \mathbf{b} \cdot \mathbf{v}_j - d \end{pmatrix}.$$

Then

$$\sum_{i<m} \frac{\partial}{\partial x_i} \mathbf{v}' = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \sum_{i<m} \frac{\partial}{\partial y_i} \mathbf{v}' = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

so we replace every $\mathbf{v}_i$ by $\mathbf{v}_i - \mathbf{v}'$ in the linear constraints. As a result, each constraint is expressed in terms of $\mathbf{v}_1 - \mathbf{v}_0, \ldots, \mathbf{v}_{k-1} - \mathbf{v}_0$. □

Theorem 2 is an immediate corollary of Theorem 26.

References

[1] H.-K. Ahn, P. Brass, and C.-S. Shin. Maximum overlap and minimum convex hull of two convex polyhedra under translations. *Comput. Geom.*, 40(2):171–177, 2008.

[2] H.-K. Ahn, S.-W. Cheng, and I. Reinbacher. Maximum overlap of convex polytopes under translation. *Comput. Geom.*, 46(5):552–565, 2013.

[3] D. Avis, P. Bose, T. C. Shermer, J. Snoeyink, G. Toussaint, and B. Zhu. On the sectional area of convex polytopes. In *Communication at the 12th Annu. ACM Sympos. Comput. Geom.*, page C. Association for Computing Machinery, New York, NY, 1996.

[4] H. Brunn. *Über Ovale und Eiflächen.* Akademische Buchdruckerei von R. Straub, 1887.

[5] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[6] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10(4):377–409, 1993.

[7] M. De Berg, O. Cheong, O. Devillers, M. Van Kreveld, and M. Teillaud. Computing the maximum overlap of two convex polygons under translations. *Theory of computing systems*, 31(5):613–628, 1998.

[8] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: sorted matrices. *SIAM Journal on computing*, 13(1):14–30, 1984.

[9] K. Fukuda and T. Uno. Polynomial time algorithms for maximizing the intersection volume of polytopes. *Pacific Journal of Optimization*, 3(1):37–52, 2007.

[10] J. Matoušek. Cutting hyperplane arrangements. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 1–9, 1990.

[11] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. Assoc. Comput. Mach.*, 31(1):114–127, 1984.

[12] N. Megiddo. Partitioning with two lines in the plane. *J. Algorithms*, 6(3):430–433, 1985.

[13] H. Minkowski. Allgemeine lehrsätze über die convexen polyeder. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1897:198–220, 1897.

[14] J. o'Rourke. *Computational geometry in C.* Cambridge university press, 1998.

[15] M. I. Shamos. *Computational geometry.* Yale University, 1978.

[16] H. Zhu and H. J. Kweon. Maximum overlap area of a convex polyhedron and a convex polygon under translation. In *39th International Symposium on Computational Geometry (SoCG 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

A Appendix

The below is the proof of Lemma 6.

Proof. By Theorem 4, we can assume that $M \neq \emptyset$. Moreover, by eliminating variables, we may also assume that $m = 0$. To compute the maximal affinely independent set, we start with an empty set S and gradually add points to it. At each step, we look for a new point that is not in the affine hull of the current set S.

To do this, we first select a linear functional h that is nonzero but evaluates to zero on all points in S. We can find such a functional in constant time since d is a constant. We then find the minimum and maximum values of h subject to the constraints in M, denoted by $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$, respectively.

If $|S| \leq \dim M$, then $h(\mathbf{x}_{\min}) < h(\mathbf{x}_{\max})$. Therefore, for some $\mathbf{x} \in \{\mathbf{x}_{\min}, \mathbf{x}_{\max}\}$, the set $S \cup \{\mathbf{x}\}$ should be also affinely independent. In this case, we replace S by $S \cup \{\mathbf{x}\}$. If not, we terminate the process. □

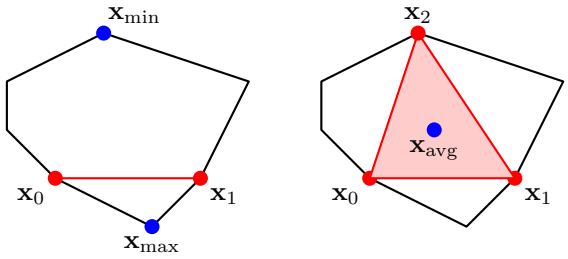


Figure 3: Finding a maximal affinely independent set and the topological interior points.

The below is the proof of Lemma 14

Proof. Let P be a polygon with n linear inequalities $f_i(\mathbf{x}) \geq 0$, sorted counterclockwise by the gradients of ∇f_i. Let ℓ_i be the line defined by $f_i = 0$, and let $v_0, \ldots, v_{m-1}$ be the vertices of Q sorted counterclockwise and indexed modulo m. Then the set of all type $(0, 1)$ lines is

$$S = \{-v_j + \ell_i \mid i < n \text{ and } j < m\}.$$

By using Lemma 13, we can determine the indices $a(i)$ and $b(i)$ for each i, such that $v_{a(i)}$ (resp. $v_{b(i)}$) is the vertex of Q that minimizes (resp. maximizes) $f_i(v_j)$ for all $j < m$. This computation can be done in $O(m + n)$-time. We can then construct two arrays:

$$A_{2i} \coloneqq (-v_{a(i)} + \ell_i, -v_{a(i)+1} + \ell_i, \ldots, -v_{b(i)-1} + \ell_i)$$

and

$$A_{2i+1} \coloneqq (-v_{b(i)} + \ell_i, -v_{b(i)+1} + \ell_i, \ldots, -v_{a(i)-1} + \ell_i),$$

whose intercepts are sorted. Note that we do not need to compute the entries of A_i explicitly; once we have computed $a(i)$ and $b(i)$, we can perform random access in $O(1)$-time using the formulas above. The resulting arrays $A_0, \ldots, A_{2n-1}$ provide a slope-intercept array representing the set of all type $(0, 1)$ lines. □

The below is the proof of Theorem 17.

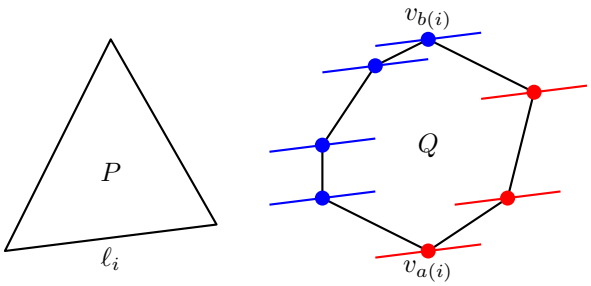


Figure 4: Visualization of why A_{2i} and A_{2i+1} are sorted.

Proof. We parameterize ℓ by

$$f(t) = (f_0(t), f_1(t), \ldots, f_{k-1}(t)),$$

where $f_i \colon \mathbb{R} \to \mathbb{R}^2$ are linear functions. We define cylinders

$$C_i \coloneqq (x, y, z) \in \mathbb{R}^3, |, (x, y) \in f_i(z) + P_i.$$

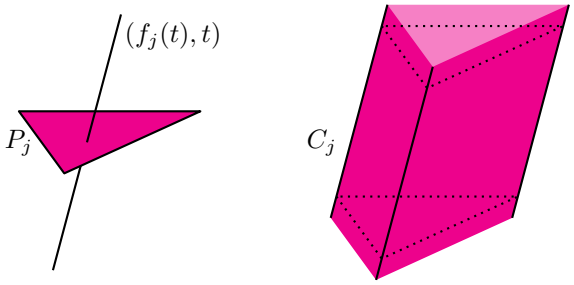


Figure 5: Depicting the cylinder C_i obtained from P_i and ℓ.

We can compute $C = C_0 \cap C_1 \cap \cdots \cap C_{k-1}$ in $O(n)$-time using Chazelle's algorithm [6]. Let $H_t \subset \mathbb{R}^3$ be the hyperplane defined by $z = t$. Then we have $|C \cap H_t| = |(P_0 + f_0(t)) \cap \cdots \cap (P_{k-1} + f_{k-1}(t))|$. We can find t maximizing $|C \cap H_t|$ in $O(n)$-time using [3, Theorem 3.2]. For such a t, the maximum point of $\Pi|_\ell$ is $f(t) \in \ell$. □

The below is the proof of Proposition 19.

Proof. The proof is similar to that of Theorem 15. Let $M \subset L$ be the set of placements maximizing $\Pi|_L$. To determine on which side of a line ℓ the set $\pi_{i,j}(M)$ lies, we apply Theorem 10, which takes $O(n \log^{r-2} n)$-time.

We can represent all type-(i, j) lines by a slope-intercept array A in $O(n)$-time, as shown in Lemma 14. Applying Theorem 12 to obtain lines ℓ_0 and ℓ_1, we can prune A to about $1/8$ of its size using Theorem 10. This step requires $O(n \log^{r-2} n)$-time. After $O(\log n)$ steps, only $O(1)$ lines remain, and then we triangulate the remaining region. This gives a triangle $T_{i,j}$ with the desired properties in $O(n \log^{r-1} n)$-time. □

The below is the proof of Lemma 24.

Proof. Since $P_0, \ldots, P_{k-1}$ are convex,

$$\frac{1}{2} I(\mathbf{u}) + \frac{1}{2} I(\mathbf{v}) \subset I\left(\frac{\mathbf{u} + \mathbf{v}}{2}\right).$$

Therefore,

$$\left|\frac{1}{2}I(\mathbf{u})+\frac{1}{2}I(\mathbf{v})\right| \leq \left|I\left(\frac{\mathbf{u}+\mathbf{v}}{2}\right)\right| \leq |I(\mathbf{v})|.$$

As a result, $I(\mathbf{u})$ and $I(\mathbf{v})$ are homothetic by Theorem 23. Since $|I(\mathbf{v})| = |I(\mathbf{u})|$, this implies that $I(\mathbf{u})$ and $I(\mathbf{v})$ are equivalent up to translation. □

B Partitioning with Two Lines

In this section, we prove Theorem 12. While the main theorems can be derived solely from [16, Theorem 1.4], this approach is somewhat unsatisfactory. Specifically, it requires three queries at every step and prunes only 1/18 of the lines, leading to a slowdown factor of 27/4. Moreover, the statement of [16, Theorem 1.4] is much more difficult to describe.

To provide a more convenient (at least in the authors' taste) proof, we instead prove the dual statement. This is the problem of partitioning a set of points in the plane with two lines such that each quadrant contains at least 1/4 of the points. We begin by presenting Megiddo's linear time algorithm for a special case of the ham sandwich problem [12, Section 2].

Theorem 27 *Given two finite sets of points in the plane with a total of n points, and with disjoint convex hulls, we can compute a line that bisects both sets in $O(n)$-time.*

The following corollary is a slightly stronger result than Megiddo's original main theorem [12].

Corollary 28 *Given a set of n points in a projective plane $\mathbb{P}^2$, we can compute a horizontal line ℓ_0 and a non-horizontal line ℓ_1 in $O(n)$-time, such that each closed quadrant defined by the two lines contains at least $\lfloor n/4 \rfloor$ points in $O(n)$-time.*

Proof. First, we can assume that there are no points on the line at infinity by applying the perturbation $(a;b;c) \mapsto (a;b;c+\varepsilon b)$. An appropriate value for ε can be computed in $O(n)$-time. Additionally, we can disregard a single point at $(1;0;0)$, as it is contained in all closed quadrants.

Next, we identify the horizontal line that passes through the median y-coordinate of the points, denoted as ℓ_0. If ℓ_0 contains at least half of the points, we can select any non-horizontal line ℓ_1 that passes through the median point m of ℓ_0. As a result, we assume that ℓ_0 contains fewer than half of the points.

We put the points above the line ℓ_0 in a set A. Moreover, we also put points on ℓ_0 from left until A has at least half of the points. Then B is the set of remaining points. Since the convex hulls of A and B are disjoint, we can apply Theorem 27 to compute the line ℓ_1 that simultaneously bisects both sets. Since ℓ_0 contains less than half of the points, ℓ_1 should not be horizontal. This divides the plane into four closed quadrants, each containing at least $\lfloor n/4 \rfloor$ points. □

An intersecting aspect is that Theorem 28 offers a linear-time algorithm for its own weighted version. It is important to note that this approach heavily relies on the following well-established result.

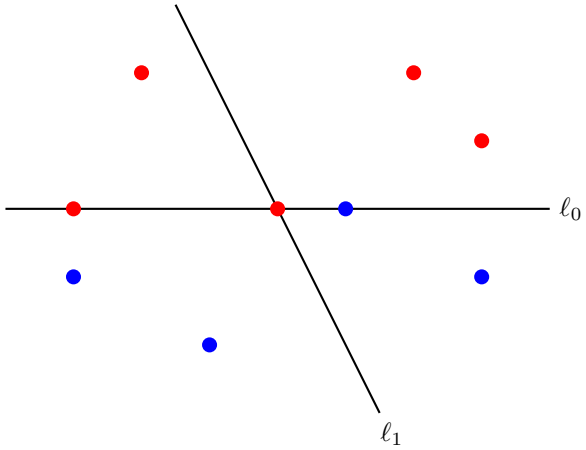


Figure 6: The red represents A and the blue represents B

Lemma 29 *Given n distinct real numbers with positive weights, we can determine the weighted median of these numbers in $O(n)$-time.*

Theorem 30 *Given n weighted points in a projective plane $\mathbb{P}^2$ with positive weights $\lambda_0, \ldots, \lambda_{n-1}$, we can compute a horizontal line ℓ_0 and a non-horizontal line ℓ_1 in $O(n)$-time such that each closed quadrant defined by the two lines contains at least 1/4 of the total weight.*

Proof. Once again, we can assume that there are no points on the line at infinity by applying perturbation $(a;b;c) \mapsto (a;b;c+\varepsilon b)$ and ignoring a single point at $(1,0,0)$. Let ℓ_0 be the weighted median horizontal line. If ℓ_0 contains at least half of the total weight, then we can choose any non-horizontal line ℓ_1 passing through the weighted median point m of ℓ_0. Therefore, we assume that ℓ_0 contains less than half of the total weight.

We start by putting all points above the line ℓ_0 into a set A, and adding points on ℓ_0 from left to right until A has at least half of the total weight. We modify the weight of the last point p so that the total weight of A is exactly half of the total weight, and set B as the remaining points and p with the remaining weight.

Since ℓ_0 contains less than half of the total weight, any ham sandwich cut of A and B must not be horizontal. We can then find two lines ℓ_0' and ℓ_1' as in Theorem 27. Let v_0 be their intersection, and let v_1 be the intersection of ℓ_1' and the line at infinity.

Without loss of generality, we may assume that the y-coordinate of ℓ_0 is at most that of ℓ_0'. We then take a line ℓ_1 passing through v_0 and bisecting the weight of B. If ℓ_1 also bisects the weight of A, then this is the desired line. Otherwise, we may assume without loss of generality that the left side of ℓ_1 contains more weight. Then any ham sandwich cut of A and B must pass through the left side of ℓ_0' with respect to v_0.

We can repeat this process with v_1. Then we determine which side of the line at infinity a ham sandwich cut of A and B must pass through with respect to v_1. After this, we identify one quadrant that does not intersect any ham

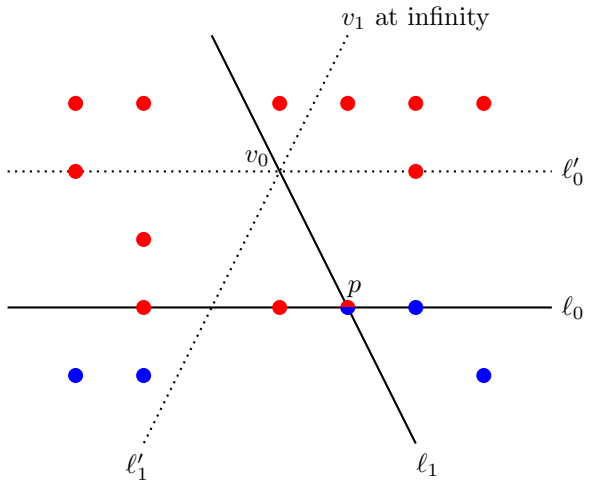


Figure 7: The red represents A and the blue represents B

sandwich cut of A and B. Thus, we can merge the points in that quadrant into two points, one for A and one for B, and repeat the entire process.

Every step, the number of points become 3/4 and we get at most 3 new points. Thus, in $O(n)$-time, at most 12 points remains. Then we can get a ham sandwich cut of A and B by brute force. The ham sandwich theorem implies that such a cut exists. □

Theorem 31 *Let A be an array of arrays $A_0, \ldots, A_{n-1}$ of points. Suppose that for each $i < m$, points on A_i lie on the same horizontal line and are sorted from left to right. Then, in $O(n)$-time, we can find ℓ_0 and ℓ_1 such that for each $i < 4$, we can obtain a pruned array P_i of A with $|P_i| \geq |A|/8$ and P_i contained in the ith quadrant.*

Proof. We can simply choose median points of each of A_i, and let the weight be the size of A_i. Then we can apply Theorem 30 and get the answer. □

Again, an intersecting aspect is that Theorem 31 offers its own optimized version.

Lemma 32 *Let S be a collection of m sorted arrays. Given x, we can compute the rank of x in $O(n \log |S|)$-time using binary search on each array.*

Proof. We can apply binear search on each array and get the answer. □

Lemma 33 *Let S be a collection of m sorted arrays. Then we can find the ith element of S in $O(n \log |S|)$-time.*

Proof. This follows from [8, Theorem 3]. □

Theorem 34 *Let A be an array of arrays $A_0, \ldots, A_{n-1}$ of points. Suppose that for each $i < m$, points on A_i lie on the same horizontal line and are sorted from left to right. Then, in $O(n \log |S|)$-time, we can find ℓ_0 and ℓ_1 such that for each $i < 4$, we can obtain a pruned array P_i of A with $|P_i| \geq |A|/4$ and P_i contained in the ith quadrant.*

Proof. The proof is almost same are that of Theorem 30. However, we need to use Theorem 31 for pruning points, Lemma 33 for bisecting B, and Lemma 32 for counting points of A. □

Let $(\mathbb{P}^2)^\vee$ be the dual projective space, which is the space parametrizing lines on $\mathbb{P}^2$. Consider the map

$$\begin{aligned} (\mathbb{P}^2)^\vee &\to \mathbb{P}^2 \\ ax + by + cz = 0 &\mapsto (c; b; a). \end{aligned}$$

Then Theorem 12 is exactly the dual theorem of Theorem 34 under this map.

Well-Separated Multiagent Path Traversal

Gleb Dilman* David Eppstein† Valentin Polishchuk‡ Christiane Schmidt§

Abstract

We consider moving points along a given path, with a fixed speed, so that no two points ever come closer than 1 (in the space into which the path is embedded, not only along the path) while they follow the path (all points traverse the path from start to finish). Since the motion of any point along the path is fully determined as soon as the point enters the path, our only decisions are the times when to send the points at the start of the path. We give algorithmic results for the problem of scheduling as many points as possible, i.e., maximizing the throughput.

1 Introduction

We study the problem of sending entities/agents along a given path so that the agents stay well separated during the motion. Such problem may arise, e.g., in an amusement park where the given path represents a ride followed by circular cabins or any entities which may deviate from the path (the path may live in 3D and the entities may represent 3D cabin volumes). Maximizing the cabin throughput maximizes the profit of the ride owner, and will also maximize the customers adrenaline, as a dense packing of the cabins implies many near misses along the ride. Similar problem appears when putting large items on a conveyor. Last but not least, the separation may be dictated by privacy or safety concerns, e.g., due to the fear of infection spread between two people or making two entities vulnerable to a single point threat/eavesdropper, which affects a certain radius around it.

For the most part, we focus on 2D; however our solutions work in arbitrary dimensions. We use realRAM model of computation – standard in computational geometry.

1.1 Related Work

To our knowledge, the considered problem was not studied before; however, a large body of work on similar questions exist:

- The *Fréchet distance* between two curves is the length of the shortest leash needed for the person on one curve to walk the dog following the other curve; computing the distance is a classical motion coordination problem (in Section 2 below, we make use of the "free space diagram" from the paper [1] that introduced Fréchet distance). More related to our problem is the recent work on *flipped* Fréchet [10], i.e., walking the dog while *maximizing* the person–dog separation. The main difference from our setup is that in both classical and flipped Fréchet settings; the person and the dog are very powerful—they can move with infinite acceleration; on the contrary, our agents move with same speed—the only decision is when to start moving (and even these starting times are not arbitrary, in some versions of our problem).
- Agents (aircraft or trains, modeled by disks and segments, resp.) following each other ducks-in-a-row along given paths were considered in the CCCG paper [16] (and also in [17]); the trains and aircraft are still more powerful than our agents because they have infinite acceleration (but their speed is bounded). Heuristics were given in [11].
- Wire routing and moving a disk through a domain (the former is hard [13] while the latter can be done in polynomial time [6,7]) is also related to our problem. Finding paths for a "snake" (the Minkowski sum of a segment and a disk, i.e., of a train and an aircraft in terms of [16]) was also studied in [13]: the problem is hard for long snakes, but is FPT-tractable w.r.t. the snake length (i.e., the problem is "length-tractable"), which is reminiscent of our results: we show that our problem is hard, but admits a PTAS if the path length is small.
- Finding separated trajectories is a well studied problem in robotics and computational geometry [2, 4, 8, 12, 14, 18, 19]; our work is different in that we do not find the agents' paths (the path—one for all agents—is given in the input, not sought in the output; our problem is purely a scheduling one).
- From non-geometric literature, remotely related to our paper is the work [5] on sets of words, avoiding a set of forbidden Hamming distance subsequences (the solution to our problem will hinge on defining forbidden intervals between the path-following agents). Another non-geometric, scheduling problem is the "pinwheel problem" in which the goal

*School of Pedagogic Skills Center, gleb.dilman@gmail.com
†Computer Science Department, the University of California, Irvine, eppstein@uci.edu
‡Communications and Transport Systems, ITN, Linköping University, valentin.polishchuk@alumni.stonybrook.edu
§Communications and Transport Systems, ITN, Linköping University, christiane.schmidt@liu.se

is to attend to a set of tasks while ensuring that each task is visited with a certain frequency: it was shown in [9] that there always exists a feasible periodic schedule – a result resembling our proof that for our problem a periodic schedule can be arbitrarily close to the optimum.
- Finding (large) gaps between agents in a periodic motion is the subject of the Lonely runner conjecture [20].

1.2 Problem Formulation and Notation

In the problem input we have a polygonal path P (possibly with self-intersections) which we call the *thread*; let n be the number of edges of P and let $\ell \in \mathbb{R}_+$ be the length of the longest edge. We treat the thread as a directed path; let s be its starting point. A *bead* is a radius-1/2 disk whose center moves with unit speed along P (starting from s). A *schedule* is a sequence $S = (t_1, t_2, \dots)$ of beads inter-release times: that is, according to S, the beads are released at s at times $0, t_1, t_1+t_2, t_1+t_2+t_3, \dots$. For convenience, we start the beads numbering from 0 (bead 0 is released at time 0).

A schedule is *feasible* if the beads never collide with each other while following P, i.e., at any time, the distance between the centers of any two beads is at least 1. The goal is to find schedules with high throughput, i.e., long-term average number of sent beads, or equivalently, to minimize the long-term average of the release times, i.e., $\lim_{m\to\infty} \sum_{j=1}^{m} t_j/m$ (the reciprocal of the throughput). We will restrict attention only to feasible schedules for which the limit exists.

Periodic schedules A schedule is *periodic* if it repeats itself, i.e., if for some p we have $t_{j+p} = t_j \ \forall j$; the minimum p for which this holds is called the *period* of the schedule. If the period $p = 1$ (i.e., if the interval between consecutive beads release is constant), the schedule is called *uniform*.

1.3 Results

Section 2, we present an $O(n^3\ell)$-time algorithm for finding optimal uniform schedules (see the paragraph above for the definiton of uniform and periodic scheudles); in Section 3, we extend the algorithm to periodic schedules of period $p = O(1)$ (the runtimes of the algorithms have p in the exponent). In Section 4, we prove that periodic schedules (with long but bounded period) are as good as arbitrary (i.e., possibly aperiodic) schedules. As a corollary we obtain that the optimal (possibly aperiodic) schedule may be approximated arbitrarily well by a periodic one, implying a PTAS for short paths. Finally, in Section 5, we prove hardness of (even approximating) the problem when the period is large.

In summary, we show that our problem is hard in general, but can be solved in polynomial time for short-period schedules; for arbitrary schedules, the problem admits a PTAS if the thread is short.

An applet to play with sending the beads (also along several paths) is available at `https://www.cs.helsinki.fi/group/compgeom/necklacegame/`: to send a bead, click on the bead at the beginning of the path. Figure 1 shows snapshots of the game.

2 Uniform Schedules

Let $t = t_1 = t_2 = \dots$ be the common value for the beads inter-release times in a uniform schedule; our goal is to minimize t. Consider two beads, with the second one following the first one at distance t along P (Fig. 2), and let e_1, e_2 be the edges of the thread on which the beads are situated at some moment in time (we do not assume that P's edges are numbered – the indicies 1 and 2 in e_1, e_2 are not ordinal numbers; in particular, it is possible that $e_1 = e_2$, or that e_1 is farther than e_2 from s). Let F_{e_1,e_2} be the set of "bad" timings t, i.e., the set of values for t that lead to collision of the beads while they are on e_1, e_2.

Lemma 1 *F_{e_1,e_2} is a single interval (possibly empty).*

Proof. Let x_1, x_2 encode the locations of the beads on e_1, e_2 resp. at some moment of time, i.e., bead i is at distance x_i from the starting point of e_i (recall that P and hence its edges are directed). Let $C \subseteq [0, |e_1|] \times [0, |e_2|]$ be the set of (x_1, x_2) pairs for which the distance between the beads is at most 1; C is the *free space* [1] for the Fréchet distance between the edges. It is well known that C is a connected subset of the (x_1, x_2)-plane (in fact, as was proved in [1], C is convex – the convexity of C follows from the convexity of the distance function). Since $t = d - x_2 + x_1$ where d be the distance (along P) from the startpoint of e_2 to the start point of e_1, the beads motion is described by a (45^o-sloped) line in the (x_1, x_2)-plane. The beads do not intersect iff the line does not intersect C, which happens for a contiguous range of t. □

For an interval $I = [a, b] \subset \mathbb{R}$ and a natural number k, let $I/k = [a/k, b/k]$ denote the "scaled down" copy of I. Having t outside F_{e_1,e_2} ensures that two consecutive beads will not collide on e_1, e_2, i.e., that for any j the bead j does not collide with bead $j+1$. To make sure that bead j does not collide with bead $j+2$, the time interval $2t$ between the beads releases should lie outside F_{e_1,e_2}, or equivalently $t \notin F_{e_1,e_2}/2$. Similarly, for the bead j to avoid bead $j+3$, it should hold that $t \notin F_{e_1,e_2}/3$. In general, to ensure no collisions of beads on e_1, e_2 we should have $t \notin F_{e_1,e_2}/k$ for any natural k. Overall, to avoid beads collisions on any pair of edges

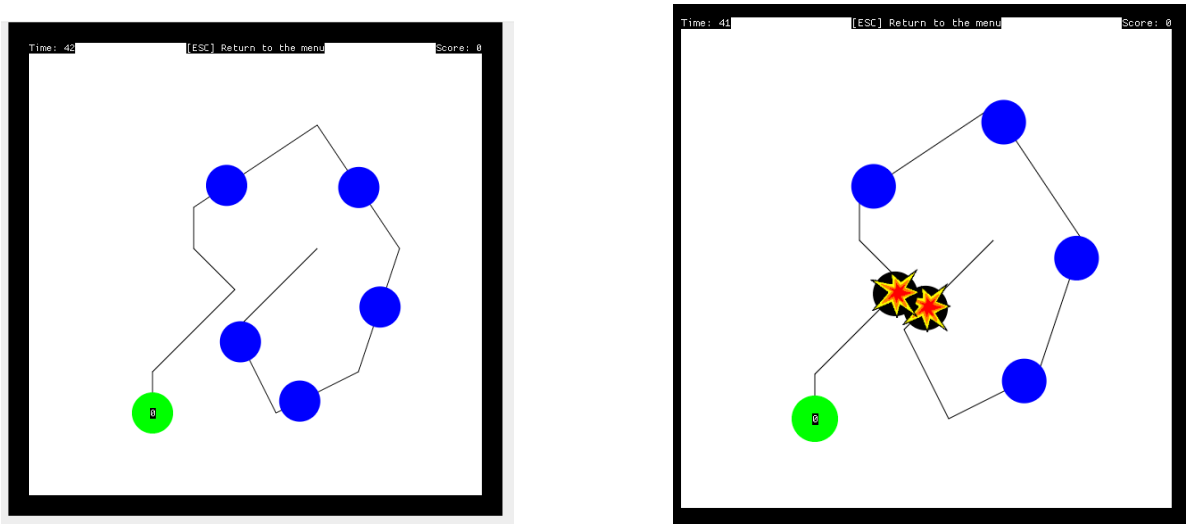


Figure 1: Left: 5 beads (blue) moving along the path. Right: a collision

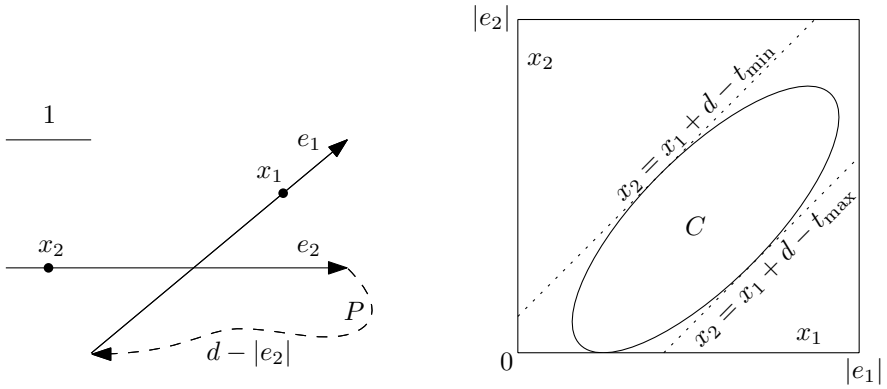


Figure 2: Left: beads at x_1, x_2 on edges e_1, e_2; the part of P between the edges (dashed) has length $d - |e_2|$. Right: $F_{e_1,e_2} = [t_{\min}, t_{\max}]$ is defined by the tangents (dotted) to C (drawn with ipelet [15]).

of the thread, t should be outside all possible forbidden intervals F_σ/k where σ ranges over all pairs of edges of P and $k \in \mathbb{N}$ (Fig. 3).

Theorem 2 *An optimal uniform schedule can be found in $O(n^3\ell)$ time.*

Proof. Let $F_\sigma = [a_\sigma, b_\sigma]$ for a pair σ of edges. It suffices to consider only those k for which $a_\sigma/k \geq 1$, since for a larger k, $[a_\sigma, b_\sigma]/k \subset [0, 1]$ and any $t < 1$ is clearly infeasible. Since the length of the thread is at most $n\ell$, any $t > n\ell$ is feasible, implying $k \leq a_\sigma \leq n\ell$. Thus, the $O(n^3\ell)$ forbidden intervals for t (over all $O(n^2)$ pairs of edges and all $k \leq n\ell$) can be constructed in time $O(n^3\ell)$. The optimal t is the smallest one not covered by the intervals: it is the left endpoint of one of the intervals. □

3 Periodic Schedules

Figure 4 gives a motivation for considering periodic schedules: they can perform arbitrarily better than uniform. We first consider schedules with period 2, defined by the two repeating beads inter-release times t_1, t_2; our goal is to minimize $t_1 + t_2$. The constraint is that no two beads ever collide—on any pair of the thread edges. Therefore, just as with uniform schedules (Section 2), for every pair σ of P's edges we compute the forbidden interval F_σ between two beads on the edges. Let $\mathcal{B} = \bigcup_\sigma F_\sigma$ be the union of all forbidden intervals, treated as a sequence of (maximal) pairwise-disjoint segments on the real line: $\mathcal{B} = [a_1, b_1] \cup [a_2, b_2] \cup \cdots \cup [a_f, b_f] \cup \ldots$ where $a_1 = 0, 1 \leq b_1 < a_2 < b_2 < a_3 \ldots$. The number of the segments is at most the number of edge pairs,

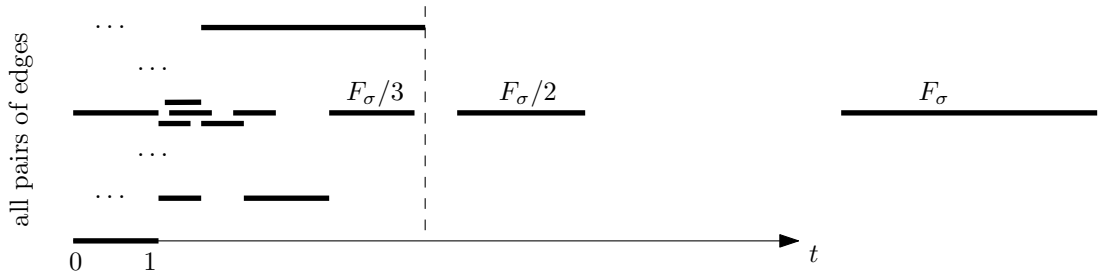


Figure 3: Optimum (dashed) is $\min t : t \notin \bigcup_{\sigma,k} F_\sigma/k$.

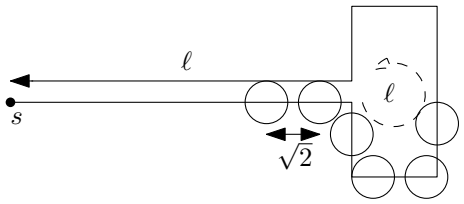


Figure 4: P is shaped as a hammer with thin handle of length $\ell \gg 1$ and the head of perimeter ℓ. A uniform schedule must have $t > \ell$ (so that beads do not collide on the opposite sides of the handle). A periodic schedule can send a length-ℓ train of $\lfloor \ell/\sqrt{2} \rfloor$ beads with inter-release times $\sqrt{2}$ (so the beads do not collide at the 90^o turns of P), wait until the train fully leaves P (time 4ℓ), and repeat. Thus the uniform schedule has throughput $\Theta(1/\ell)$ while the throughput of the periodic schedule is constant (the long-term average of the release times is $> \ell$ for uniform schedules and $\Theta(1)$ for a periodic one).

$O(n^2)$.

In a period-2 schedule, the interval (the distance along the thread) between beads of the same parity is $k(t_1+t_2)$ for an integer $k \geq 1$. The interval between beads of different parity is either $k(t_1 + t_2) + t_1$ (from an even bead to an odd) or $k(t_1 + t_2) + t_2$ (from an odd to an even bead) for $k \geq 0$. Thus for a feasible schedule it is necessary and sufficient that

$$\begin{array}{cccc} (k+1)(t_1+t_2), & k(t_1+t_2)+t_1, \\ k(t_1+t_2)+t_2 & \notin & \mathcal{B} & \forall k = 0, 1, \ldots, n\ell/2 \end{array} \quad (1)$$

(as with uniform schedules, $t_1, t_2 \geq 1$, and it suffices to consider inter-release times that do not exceed the maximum thread length, $n\ell$, i.e., $(k+1)(1+1) \leq n\ell$).

For any one forbidden segment $[a_f, b_f]$ from $\mathcal{B}$ and any fixed k, the inequalities $a_f \leq (k+1)(t_1+t_2) \leq b_f$ define a slab of forbidden pairs in the (t_1, t_2)-plane (Fig. 5). Similarly, the inequalities $a_f \leq k(t_1+t_2)+t_1 \leq b_f$ and $a_f \leq k(t_1+t_2)+t_2 \leq b_f$ each define a slab. Overall, i.e., for segments $[a_f, b_f]$ for all $O(n^2)$ f's and $O(n\ell)$ k's, the requirements (1) define $O(n^3\ell)$ slabs. We build the arrangement of the slabs and find the vertex of the arrangement minimizing $t_1 + t_2$ in $O(n^6\ell^2)$ time

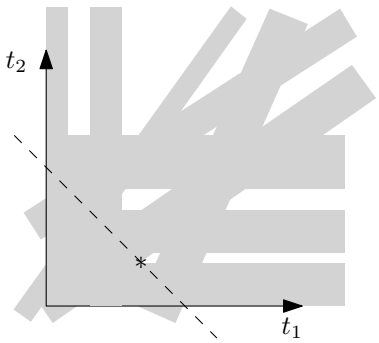


Figure 5: The requirements (1) define slabs of forbidden (t_1, t_2) pairs (gray). The optimal schedule (marked with the asterisk) minimizes $t_1 + t_2$ for points outside the slabs (white).

(by going through all the vertices).

The above algorithm extends to schedules of any length p:

Theorem 3 *An optimal schedule with period p can be found in $O((n^3 p\ell)^p)$ time.*

Proof. Let $b, b' \in \mathbb{Z}_0^+, b < b'$ be two beads, identified with their positions in the schedule (recall that we start numbering the beads from 0). If $b \equiv b' \bmod p$, then in a period-p schedule the interval (the distance along the thread) between the beads is $(k+1)(t_1 + t_2 + \cdots + t_p)$ for an integer $k \geq 0$. More generally, if $b \equiv r \bmod p$ and $b' \equiv r' \bmod p$, the distance is

$$\begin{array}{c} D(r.r', k) = t_{r'+1} + t_{r'+2} + \cdots + t_p + \\ +k(t_1 + \cdots + t_p) + t_1 + t_2 + \cdots + t_r \end{array} \quad (2)$$

Thus for a feasible schedule it is necessary and sufficient that

$$\begin{array}{cc} & D(r, r', k) \notin \mathcal{B} \\ \forall k = 0, 1, \ldots, n\ell/p, & \forall r, r' = 0, 1, \ldots p-1 \end{array} \quad (3)$$

For any one segment $[a_f, b_f]$ from $\mathcal{B}$ and any fixed r, r', k the inequalities $a_f \leq D(r, r', k) \leq b_f$ define a slab of forbidden schedules in the $(t_1, t_2, \ldots, t_p)$-space—overall, i.e., for segments $[a_f, b_f]$ for all $O(n^2)$ f's,

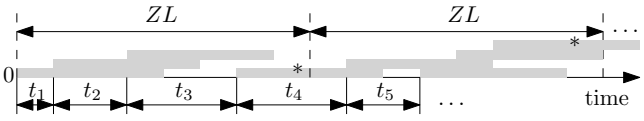


Figure 6: Length-L segments (gray) correspond to beads in the maximum-throughput schedule; we will remove the segments marked with asterisks.

$O(n\ell/p)$ k's and $O(p^2)$ pairs (r, r'), the requirements (1) define $O(n^3 p\ell)$ slabs in the p-dimensional space. We build the arrangement of the slabs and find the vertex of the arrangement minimizing $t_1 + t_2 + \cdots + t_p$. □

4 Arbitrary Schedules

Let τ be the value of optimal throughput over arbitrary, possibly aperiodic schedules. We show that periodic schedules can achieve a throughput arbitrarily close to τ:

Theorem 4 *For any $\varepsilon > 0$ there exists a periodic schedule whose throughput is at least $\tau - \varepsilon$.*

Proof. Let L denote the length of P. Identify each bead with the length-L segment on the real line, spanning the time during which the (center of) the bead traverses P (Fig. 6). Consider the segments in the optimal (possibly aperiodic) schedule S. Choose an integer $Z > 2L\tau^2/\varepsilon$ and draw vertical lines $t = ZL, t = 2ZL, \ldots$ at the regular spacing ZL. We claim that no such line intersects the interior of more than $x = 2L\tau$ segments. Indeed, the segments intersecting any one line cover at most $2L$ of the time. If there existed a line intersecting a set of more than x segments, we could have laid such sets one after another, obtaining a (periodic) schedule with throughput at least $x/2L > \tau$, contradicting the global optimality of S.

Now remove from S the segments whose interior is intersected by one of the drawn vertical lines. By the above, the fraction of the removed segments is less than x/Z, implying that the throughput of the schedule with the remaining segments is greater than $\tau - \tau x/Z > \tau - \varepsilon$. It follows that at least one set of segments between the lines has throughput $> \tau = \varepsilon$; (in fact, such sets should appear infinitely often, but for us it is enough to) take one such set and repeat it – this results in a periodic schedule with throughput $> \tau - \varepsilon$. □

It follows from the proof of Theorem 4 that to get a schedule with throughput $\tau(1 - x/Z) > \tau(1 - \varepsilon/\tau)$ it is enough to find an optimal schedule in a length-ZL interval. Since the number of beads sent during the interval is $O(ZL) = O(L^2\tau^2/\varepsilon)$, it suffices to consider schedules with period $O(L^2\tau^2/\varepsilon)$. Taking $\delta = \varepsilon/\tau$, we conclude that a $(1-\delta)$-approximation to the maximum throughput can be obtained by considering schedules with period $p = O(L^2\tau/\delta) = O(L^3/\delta)$ since $\tau < L$. From Theorem 3,

Theorem 5 *A $(1-\delta)$-approximation to the maximum throughput can be found in $O((nL/\delta)^{O(L^3/\delta)})$ time.*

In particular, for threads with constant length, our problem has a PTAS.

5 Hardness of Approximating Arbitrary Schedules

Our result is based on a known inapproximability results for maximum clique by Arora et al. [3]:

Theorem 6 (Arora et al. [3]) *There is a constant $c > 0$, such that approximating the maximum clique size in an N-vertex graph to within a factor N^c is NP-hard.*

We combine this result for the maximum clique problem with an approximation-preserving reduction from maximum clique to our problem, that is, the problem of determining the optimal schedule of beads. Hence, from a given graph G in which we aim to find a maximum clique we will construct a thread P, such that the optimal release times for P correspond to a maximum clique in G. Thus, the approximation ratio for the optimal schedule of beads for P cannot be better than the approximation ratio for maximum clique in G (given by Theorem 6).

For the construction of P from G, we define a set $\mathcal{I}$ of maximal intervals of the timeline such that, if a bead is released at time 0, it is safe to release another bead within one of these intervals (and inter-release times $\sum_{j=i}^{i+k} t_j \notin \mathcal{I}, i = 1, 2, \ldots; k = 0, 1, \ldots$ are infeasible). We call $\mathcal{I}$ the set of *safe* intervals. Assuming that we can construct $\mathcal{I}$ as desired, we describe the approximation-preserving reduction from maximum clique in Section 5.1, and we detail the construction of $\mathcal{I}$ in Section 5.2.

5.1 Reduction from Maximum Clique

Let G be the graph in which we want to solve the maximum clique problem, let $|V(G)| = N$, and let the vertices be labeled $1, \ldots, N$. We use a greedy algorithm to construct a set $U = \{u_1, u_2, \ldots u_N, u_{N+1}\}$ of integers, such that $u_{j1} \pm u_{j2} \pm u_{j3} \pm u_{j4} \pm u_{j5} \pm u_{j6} \neq 0$ for any 6 indicies $j_1, \ldots, j_6 \in \{1, \ldots, N+1\}$. We start with $u_1 = 2$ (see Section 5.2 on why we do not start with 1) and consider integers of increasing value, adding them to U whenever they do not yield an infeasible linear combination with the numbers previously added to U. For integers $1, \ldots, k$, we need to pick at least $k^{1/5}$ integers for U in order to exclude the other numbers due to infeasible linear combinations. Thus, $u_N \in O(N^5)$.

We construct a thread P with safe intervals around $\{u_i \mid 1 \le i \le N\} \cup \{u_i - u_j \mid i > j \text{ and } ij \in E(G)\}$ and a semi-infinite interval starting at u_{N+1}. That is

$$\mathcal{I} = \bigcup_{i=1}^{N} (u_i - \varepsilon, u_i + \varepsilon) \cup \bigcup_{ij \in E(G); i > j} (u_i - u_j - \varepsilon, u_i - u_j + \varepsilon) \cup [u_{N+1}, \infty), \quad \varepsilon > 0 \tag{4}$$

Any optimal schedule with release times smaller than u_{N+1} can be repeated at intervals of time u_{N+1}. Hence, we obtain a finite problem: Find the largest subset of the finite parts of $\mathcal{I}$ all of whose differences are in $\mathcal{I}$.

Let $K = \{v_{k_1}, v_{k_2}, \ldots, v_{k_{|K|}}\}$ be the set of vertices in a clique with $k_1 < k_2 < ... < k_{|K|}$, then release times $0, t_1, t_1 + t_2, \ldots$ with

$$\sum_{i=1}^{j} t_i = u_{k_j}, j = 1, \ldots, |K| \tag{5}$$

yield a feasible schedule of $|K| + 1$ release times: the inter-release times (for $j_1 < j_2$) are $(t_1 + t_2 + \ldots + t_{j_2}) - (t_1 + t_2 + \ldots + t_{j_1}) = u_{k_{j_2}} - u_{k_{j_1}} \in \mathcal{I}$, because $(v_{k_{j_1}}, v_{k_{j_2}}) \in E(G)$. Moreover, any feasible schedule for P for which all release times are 0 or elements of U must be of this form.

Additionally, for a clique K in G the set $\{0, u_{k_{|K|}}\} \cup_{j=1,\ldots,k_{|K|-1}} \{u_{k_{|K|}} - u_j\}$ is a feasible set of release times. There also exist feasible schedules with release times of the form $u_{k_i} - u_{k_j}$. The release time $u_{k_i} - u_{k_j}$ is compatible with u_{k_i} (because $u_{k_i} - (u_{k_i} - u_{k_j}) = u_{k_j} \in \mathcal{I}$) and with a single other time $u_{k_j} - u_{k_{j^*}}$ (because $(u_{k_i} - u_{k_j}) - (u_{k_j} - u_{k_{j^*}}) = u_{k_i} - u_{k_{j^*}} \in \mathcal{I}$ if $ij^* \in E(G)$), but not with more release times of this form. Hence, the schedules that do not stem from cliques in G have bounded size. Thus, if we aim for cliques larger than that, the release times in an optimal schedule must stem from a clique in G.

Using Section 5.2, we construct a thread P with $n_P = 3 + 7 \cdot N + 7 \cdot |E(G)|$ edges with $|\mathcal{I}| = f(n_P) = N + |E(G)| + 1$ safe intervals. With Theorem 6 we yield:

Theorem 7 *There exists a constant $c > 0$, such that approximating the optimal schedule of beads in a thread with n_P edges to within a factor $n_P^{1/2-c}$ is NP-hard.*

Proof. Suppose there exists an algorithm $\mathcal{A}$ that can schedule beads optimally within a factor $\rho \le n_P^{1/2-c}$. Then we can construct an algorithm $\mathcal{A}'$ for the maximum clique problem in G:

1. Use Lemma 8 to construct P from the given graph G (the input for the maximum clique problem).
2. Use algorithm $\mathcal{A}$ to schedule beads on P.
3. Construct a maximum clique for G from the release times for P.

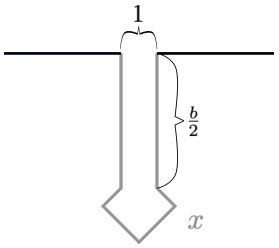


Figure 7: Gadget to exclude the interval $[x-b, x]$. The black lines are the horizontal edges of P adjacent to the gadget. The total length of the (gray) gadget is x.

Then the approximation ratio for $\mathcal{A}'$ for the maximum clique problem is the same as for algorithm $\mathcal{A}$ for scheduling beads (ρ). Hence, we have:

$$\rho \le n_P^{1/2-c} = O((N+M)^{1/2-c}) = (N^{1-2c}) \tag{6}$$

which yields a contradiction to Theorem 6. □

5.2 Construction of the Set of Safe Intervals

We aim to exclude all but the set of safe intervals given in Equation (4). We use a long horizontal path to which we add several gadgets to exclude certain intervals. Between each pair of consecutive gadgets we have a horizontal edge of length u_{N+1} in P:

- To exclude $(0, u_1 - \varepsilon]$ use a path of length $u_1 - \varepsilon$ that runs on itself, i.e., $\frac{u_1-\varepsilon}{2}$ up and $\frac{u_1-\varepsilon}{2}$ down.
- To exclude intervals of the form $[x - b, x]$ we use the gadget shown in Figure 7: The total length of the gadget (shown in gray) is x, we have two vertical edges of length $\frac{b}{2}$ each within a distance of 1, and part of a slanted square with four edges, total length $x - b$, and a distance > 1 between parallel edges. For example, if we aim to exclude the interval $[u_i + \varepsilon, u_{i+1} - \varepsilon]$ from the safe intervals, we choose $x = u_{i+1} - \varepsilon$, $b/2 = u_i - \varepsilon$, which excludes the interval $[x-b, x]$ with $x - b = u_{i+1} - \varepsilon - 2*(u_i - \varepsilon) = u_i + \varepsilon$. Each of these gadgets also excludes the interval $(0, \sqrt{2}]$, thus, we choose $u_1 > 1$.

Each gadget except for the first has one horizontal edge and six edges within the gadget. The gadget excluding the interval $(0, u_1 - \varepsilon]$ has three edges. Hence, for a graph G with N vertices, constructing the set $\mathcal{I}$ of safe intervals from Equation (4), that is, excluding all "unsafe" intervals, we use $n = 3 + 7 \cdot N + 7 \cdot |E(G)|$ edges. This yields:

Lemma 8 *Given a graph G with $|V(G)| = N, |E(G)| = M$, we can construct in polynomial time a thread P with $n_P = 3 + 7 \cdot N + 7 \cdot |E(G)|$ vertices, and $C(G) \le S(P) \le C(G) + 1$, where $C(G)$ is the maximum clique size in G, and $S(P)$ the length of an optimum finite schedule for P.*

Acknowledgements We thank the anonymous reviewers for their helpful comments. This research is partially supported by the Swedish Transport Administration and the Swedish Research Council.

6 Conclusion

We gave pseudopolynomial-time algorithms and hardness results for scheduling uniform motion of well separated agents along a given path; our algorithms extend to the case of agents following multiple (constant number of) paths. An open problem is the existence of a polynomial-time solution.

References

[1] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.

[2] E. M. Arkin, J. S. Mitchell, and V. Polishchuk. Maximum thick paths in static and dynamic environments. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 20–27, 2008.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998.

[4] A. T. Becker, S. P. Fekete, P. Keldenich, M. Konitzny, L. Lin, and C. Scheffer. Coordinated motion planning: The video (multimedia exposition). In *34th International Symposium on Computational Geometry (SoCG 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018.

[5] V. D. Blondel, R. Jungers, and V. Protasov. On the complexity of computing the capacity of codes that avoid forbidden difference patterns. *IEEE Transactions on Information Theory*, 52(11):5122–5127, 2006.

[6] D. Z. Chen and H. Wang. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms (TALG)*, 11(4):1–46, 2015.

[7] L. P. Chew. Planning the shortest path for a disc in o (n 2log n) time. In *Proceedings of the first annual symposium on Computational geometry*, pages 214–220, 1985.

[8] E. D. Demaine, S. P. Fekete, P. Keldenich, H. Meijer, and C. Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019.

[9] E. A. Feinberg and M. T. Curry. Generalized pinwheel problem. *Mathematical Methods of Operations Research*, 62:99–122, 2005.

[10] O. Filtser, M. Goswami, J. S. Mitchell, and V. Polishchuk. On Flipping the Fréchet distance. In *ITCS - Innovations in Theoretical Computer Science*, 2023.

[11] J. Kim, A. Kröller, and J. Mitchell. Scheduling aircraft to reduce controller workload. In *9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)(2009)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2009.

[12] Y. Kobayashi and K. Otsuki. Max-flow min-cut theorem and faster algorithms in a circular disk failure model. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1635–1643. IEEE, 2014.

[13] I. Kostitsyna and V. Polishchuk. Simple wriggling is hard unless you are a fat hippo. *Theory of Computing Systems*, 50(1):93–110, 2012. Special issue on FUN'10.

[14] S. Neumayer, A. Efrat, and E. Modiano. Geographic max-flow and min-cut under a circular disk failure model. *Computer Networks*, 77:117–127, 2015.

[15] G. Röte. Free-space diagram ipelet. `https://www.mi.fu-berlin.de/inf/groups/ag-ti/software/ipelets.html`.

[16] C. Scheffer. Scheduling Three Trains is NP-Complete. In *CCCG*, pages 87–93, 2020.

[17] C. Scheffer. Train scheduling: Hardness and algorithms. In *WALCOM: Algorithms and Computation: 14th International Conference, WALCOM 2020, Singapore, Singapore, March 31–April 2, 2020, Proceedings 14*, pages 342–347. Springer, 2020.

[18] A. Sen, S. Murthy, and S. Banerjee. Region-based connectivity-a new paradigm for design of fault-tolerant networks. In *2009 International Conference on High Performance Switching and Routing*, pages 1–7. IEEE, 2009.

[19] A. Sen, B. H. Shen, L. Zhou, and B. Hao. Fault-tolerance in sensor networks: A new evaluation metric. In *INFOCOM 2006: 25th IEEE International Conference on Computer Communications*, page 4146923, 2006.

[20] T. Tao. Some remarks on the lonely runner conjecture. *arXiv preprint arXiv:1701.02048*, 2017.

Carving Polytopes with Saws in 3D

Eliot W. Robson* Jack Spalding-Jamieson† Da Wei Zheng‡

Abstract

We investigate the problem of *carving* an n-face triangulated three-dimensional polytope using a tool to make cuts modelled by either a half-plane or sweeps from an infinite ray. In the case of half-planes cuts, we present a deterministic algorithm running in $O(n^2)$ time and a randomized algorithm running in $O(n^{3/2+\varepsilon})$ expected time for any $\varepsilon > 0$. In the case of cuts defined by sweeps of infinite rays, we present an algorithm running in $O(n^5)$ time.

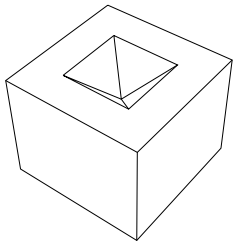


Figure 1: A polytope containing a cavity that can be carved with ray sweeps but not half-planes.

1 Introduction

Stone carving is one of the earliest known representational works of art, and has been known to predate even the earliest human civilization. This is the practice of taking a single solid piece of material and removing pieces until achieving a desired final shape. To ensure durability of the finished product, the base material is often very durable and can be difficult to carve with tools. As a result, it is desirable to minimize the amount of work that must be done to achieve the final carving, and is useful to be able to determine what kind of objects can be carved out with the tools being used.

1.1 2D Cutting

The two-dimensional case of cutting material was first studied by Overmar and Welzl [13]. In this work, the authors modeled cuts as lines in the plane, giving algorithms for computing the cheapest sequence of cuts in special cases. This was generalized by Demaine, Demaine, and Kaplan [8] to the case of cutting with line segments in the plane where they gave an algorithm with 2.5 approximation factor. This approximation factor was later improved by Dumitrescu and Hasan [9]. In 2009, Bereg, Daescu, and Jiang [3] presented a PTAS for the problem of minimum length when cutting convex n-gons out of convex m-gons with straight line cuts.

An analogous problem has been studied for ray cuts [6, 14], where the cutting object is a ray instead of a line or line segment. These results focus on carving both convex and simple polygons, and minimizing the length of cuts. In Section 3, we study a more general version of the decision variant of this problem, where we find maximum ray-carveable regions not crossing a set of disjoint polygons.

1.2 3D Cutting

Surprisingly, we found very little work on 3D generalizations of these problems, the only ones being guillotine cuts (cuts that go all the way through) used to cut a convex polygon out of a sphere [2], and work on cutting styrofoam with hot wire [12].

We explore carving three-dimensional shapes with models of two different classes of cuts:

- Straight-cuts with tools like circular saws or table saws, modelled using half-plane cuts (see Figure 3).
- Tools with the ability to pierce up to a specific depth such as waterjets or laser cutters, modelled using ray sweeps (see Figure 1).

Note that straight-cuts can also be performed with a wide variety of other tools, such as band saws, the long edge of chainsaws, or even jigsaws. Some tools also allow for additional types of cuts that we do not model, such as using the tip of a chainsaw, or rounded cuts using a band saw. However, straight-cuts can be performed with every sufficiently large saw, and moreover they are the only useful type of cut for carving polytopes exactly, since polytopes do not have rounded edges.

For simplicity, we only consider 3D polytopes P with n vertices. We make no assumptions of general position in this work, although we only consider polytopes without self-intersections or degenerate faces. In other words, we only consider polytopes that are uniquely defined by their connected interiors.

We will assume we are always given a polytope P

*Department of Computer Science, University of Illinois Urbana-Champaign, erobson2@illinois.edu

†David R. Cheriton School of Computer Science, University of Waterloo, jacksj@uwaterloo.ca

‡Department of Computer Science, University of Illinois Urbana-Champaign, dwzheng2@illinois.edu

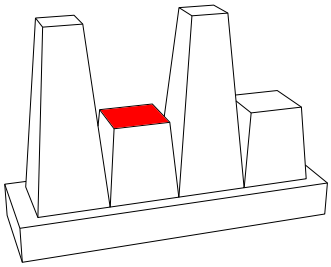


Figure 2: A polytope that can be carved with rays, but with exactly one face that cannot be carved by half-plane cuts.

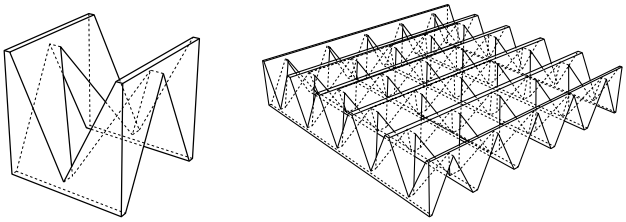


Figure 3: Non-convex polytopes with holes that can be carved using half-planes.

along with a triangulation of each face. If not, we may triangulate each face in linear time [4] with no impact on the final running time. We henceforth refer to the resulting triangles as the facial triangles. We will use n denote the complexity of P, i.e. the number of facial triangles. Note that this is asymptotically proportional to the number of vertices of P.

1.3 Our Contributions

In our paper, we consider a polytope P and determine if there exists a finite set of cuts K such that for any set $C \supset P$, the "carved" set $C \setminus K$ has a connected component (i.e., maximal connected open subset) equal to $\text{int}(P)$ (the *interior* of P). Essentially, the excess connected components ("material") can be "removed" to leave only the intended shape. We also wish this to be independent of C (i.e. C is not given as input to the algorithm) so that the produced set of cuts can be used to carve an object, regardless of the initial material.

In Section 2 we consider the set of cuts to be half-planes, which can include complicated polytopes with holes as in Figure 3. We are able to characterize the shapes that can or cannot be carved, and use this to derive a simple deterministic algorithm that runs in $O(n^2)$ time. Furthermore, we also discuss a randomized algorithm that runs in $O(n^{3/2+\varepsilon})$ time for any $\varepsilon > 0$.

In Section 3, we model ray cuts as a finite set of *ray sweeps* that consist of bounded continuous movement of a ray on a plane. Sweeps are meant to model cuts one could perform or program a machine to perform, rather than simply being the set of shapes that are excluded by a infinite union of rays. This model allows for a more general class of cuts than half-plane cuts, as in Figure 1 and Figure 2. We are able to characterize the shapes that can be carved with ray sweeps, and in this case we present an algorithm that runs in $O(n^5)$ time.

2 Half-Plane Cuts

In this section, we study half-plane cuts which model cuts that can be performed with tools such as circular saws or table saws. Formally, a half-plane is defined by a plane in $\mathbb{R}^3$ and the region on one side of a line contained in that plane. A polytope P is *carveable* if there is a set of half-plane cuts $\mathcal{H}$, so that for any P-containing set $C \supset P$, the set $C \setminus (\cap_{H \in \mathcal{H}} H)$ has a connected component (i.e., maximal connected open subset) equal to $\text{int}(P)$ (the *interior* of P).

This model is equivalent to that of an open question posed by Demaine et al. [8, 7th open problem]. They ask if there exists an algorithm to cut three dimensional polyhedra using an infinitely long rectangle that can only slice straight.

2.1 Characterization of Carveable Polytopes

For a set S, we denote the convex hull of S by $\text{CH}(S)$. We give a complete characterization of the polytopes P which can be carved in this model.

Theorem 1 *For a triangulated polytope P, the following are equivalent:*
(a) P is carveable.
(b) For each facial triangle T in P, there is a single half-plane H_T containing T such that $H_T \cap \text{int}(P) = \varnothing$.
(c) For each facial triangle T in P, let L_T denote the plane containing T. There is a single half-plane H containing T and not containing $\text{CH}\big(\text{int}(P) \cap L_T\big)$ whose boundary line passes through a vertex of $\text{CH}\big(\text{int}(P) \cap L_T\big)$, and also passes through a vertex v of T.

Essentially, the transformation from (a) to (b) indicates that it suffices to consider one cut per facial triangle, and the transformation from (b) to (c) indicates that it suffices to consider only a limited set of potential cuts for each facial triangle.

Proof.
$(a) \Rightarrow (b)$ Assume that P can be carved by a set of half-planes W. For a facial triangle T lying on the plane L_T, let $W_T \subset W$ be the subset of half-planes on L_T. Then, by definition, $T \subset \bigcup_{w \in W_T} w$. We claim that there exists a half-plane $H \supset T$ such that $H \cap \text{int}(P) = \varnothing$. Then $\bigcup_{w \in W_T} w$ is the complement of an open convex set on L_T. Denote this open convex set

on L_T as S. Note that S contains $\text{int}(P) \cap L_T$, since each $w \in W_T$ does not intersect $\text{int}(P)$. As S and T are disjoint convex sets, there exists a separating line l between S and T on L_T, which induces a half-plane H with boundary l containing T satisfying $H \cap \text{int}(P) = \varnothing$, as desired.

$(b) \Rightarrow (a)$ Take all f cuts of the form H_T. This divides the exterior of P in any P-containing set C, and does not intersect P, leaving it as a connected component.

$(b) \Rightarrow (c)$ Let T be a facial triangle of P, and let L_T denote the plane containing T. Assume that $L_T \cap \text{int}(P) \neq \varnothing$. Let H_T be a half-plane on L_T containing T and not intersecting P, i.e. $H_T \cap T = T$ and $H_T \cap \text{int}(P) = \varnothing$. A half-plane $H'_T \supset H_T$ can be found such that H'_T touches the boundary of $\text{CH}\left(\text{int}(P) \cap L_T\right)$ by translating H_T. Another half-plane H''_T can be found by rotating H'_T around the boundary of $\text{CH}\left(\text{int}(P) \cap L_T\right)$ until the result touches a vertex of T. H''_T is then tangent to $\text{CH}\left(\text{int}(P) \cap L_T\right)$, and some vertex v of T is on its boundary.

$(c) \Rightarrow (b)$ Trivial. □

We now present observations about facial triangles T and $\text{CH}\left(\text{int}(P) \cap L_T\right)$, where L_T is the plane on which T lies. The first is stated in somewhat greater generality.

Observation 1 *Let L be a plane, and let E be the set of line segments defining edges of P that cross L. That is, each line segment $e \in E$ has two endpoints which are strictly separated by L (i.e. the endpoints of e lie on opposite sides of L). Then,* $\text{CH}\left(\text{int}(P) \cap L\right) = \text{CH}(E \cap L)$.

This is because the boundary of $\text{int}(P) \cap L$ consists of vertices that defined by the intersections of E with L.

We make an additional observation about polytopes that are not carveable.

Observation 2 *Let v be a vertex of P on facial triangle T contained in the plane L_T. If there exists three edges e_1, e_2, e_3 of P such that $v \in \text{CH}\left(\{e_1, e_2, e_3\} \cap L_T\right)$, then the polytope P is not carveable.*

This directly follows from Theorem 1(c), as $\text{CH}\left(\{e_1, e_2, e_3\} \cap L_T\right) \subseteq \text{CH}\left(\text{int}(P) \cap L_T\right)$.

2.2 Quadratic Time Decision Algorithm

Using the characterization from Theorem 1(c) and the observations, we show a simple quadratic time algorithm exists for determining if a polytope is carveable.

Theorem 2 *Given a triangulated polytope P with f facial triangles, there is an algorithm to determine if P can be carved by half-planes that runs in $O(n^2)$ time. If the answer is in the affirmative, the algorithm also outputs a set of n half-planes that carve P.*

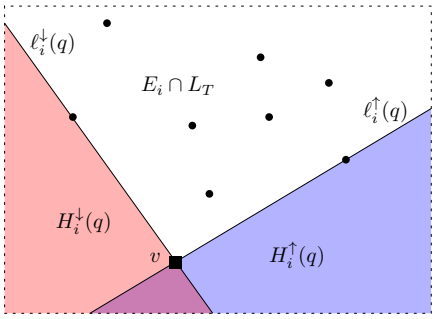


Figure 4: The tangents of v and the halfplanes $H_i^{\uparrow}(v)$ and $H_i^{\downarrow}(v)$. The figure is drawn on the plane L_T.

Proof. Let T be a facial triangle, and let L_T be the plane containing T. By Observation 1, we can compute the set of edges E of P that cross L to compute $\text{CH}\left(\text{int}(P) \cap L_T\right)$ in $O(n \log n)$ total time. If the region is empty, then any half-plane containing T can be output. Otherwise, by characterization (c) from Theorem 1, we must determine if there exists a half-plane H containing T and tangent to $\text{CH}\left(\text{int}(P) \cap L_T\right)$, such that the boundary line of H passes through a vertex v of T. Note that each vertex v of T induces up to two lines going through v and tangent to $\text{CH}\left(\text{int}(P) \cap L_T\right)$. It suffices to consider each of them. It is possible to compute tangents in $O(\log n)$ time per vertex once we have explicitly computed the convex hull.

However, we can shave the log factor (pun intended), and do this in $O(n)$ time per facial triangle with a different algorithm. The tangent to $\text{CH}\left(\text{int}(P) \cap L_T\right)$ on L_T through a vertex v is a line through a vertex v and a vertex of $X = E \cap L_T$. Let $l_{x,v}$ be the line passing through a point x in X and a vertex v of T. There are at most $O(n)$ such lines since $|E| = O(n)$. If $l_{x,v}$ is a separating line w.r.t. the plane L_T between $X \setminus \{x\}$ and the other two vertices of T, then the half-plane H containing T with boundary $l_{x,v}$ satisfies all of our conditions. Otherwise, if no such line exists, then no half-plane satisfying our conditions exists, and P cannot be carved using half-planes by Theorem 1.

It is not immediately clear how we can efficiently check each of these lines. However, observe that only the "extreme" lines $l_{x,v}$ for any specific v can be candidate separating lines. This can be accomplished by performing what is essentially an iteration of the gift wrapping algorithm for convex hulls [7, Section 1.1]. Maintain a "current" line $l_{x',v}$. Iterate through all the points $x \in X$. If x lies clockwise (resp. counter-clockwise) relative to the ray from v to x', set $x' \leftarrow x$.

If v lies outside of $\text{CH}(X)$, this procedure is guaranteed to find the tangents to $\text{CH}(X)$ passing through v as candidate lines, and we can check in linear time whether the candidate lines separate T and X. If v lies within $\text{CH}(X)$, then no such line exists, so we conclude

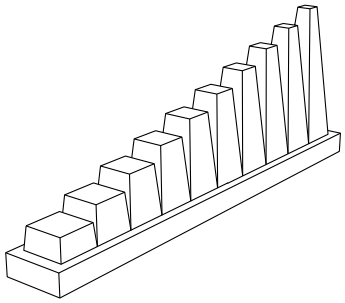


Figure 5: A construction for family of polytopes carveable using half-planes (to increase the size, add more pillars). For a member of this family with n facial triangles, the algorithm described in Theorem 2 takes $\Theta(n^2)$ time.

that the polytope P is not carveable.

This algorithm shows how to find a half-plane for each facial triangle in $O(n)$ time or deduce one doesn't exist. The whole algorithm runs in $O(n^2)$ time, since there are $O(n)$ facial triangles. □

Figure 5 demonstrates that there exists a family of polytopes that can be carved using half-planes, for which the runtime of the algorithm described in Theorem 2 is $\Theta(n^2)$. In the next section we show a faster algorithm for deciding if a polytope is carveable.

2.3 Subquadratic Time Decision Algorithm

At a high level, our quadratic time algorithm iterates through a set $\mathcal{Q}$ of queries consisting of pairs $q = (v, T)$, where v is a vertex on a facial triangle T in the polytope. Each query $q = (v, T)$ asks for tangents of $\mathrm{CH}(E \cap L_T)$ that go through v, lying on the plane L_T that contains T. Since we have $|\mathcal{Q}| = O(n)$ queries and the polygon we are querying can be different for different planes L_T and can also itself have size $O(n)$, this seems to require quadratic time. However, we note that the polygons are related: They all come from the original polytope P. Surprisingly, we show that we can solve this problem faster by answering all of these queries simultaneously.

We present a reduction from the decision problem of whether a polytope P is carveable using half-planes to the problem of detecting intersections between half-planes and line segments in $\mathbb{R}^3$. The reductions solves each query of $\mathcal{Q}$ on an increasingly large random subset of the edges E that define the polytope P, by computing the violations between half-planes induced by the previous tangents, and using the randomized analysis of Clarkson–Shor [5] to bound the number of violations. The violations are found by solving a problem involving intersections between half-planes and line segments.

Lemma 3 *Let $T(n, k)$ be the time complexity of an algorithm for detecting intersection between n half-planes and n line segments in $\mathbb{R}^3$ with at most k intersections. Then, for any parameter $2 \le r < n$, there is a randomized algorithm that decides if a polytope P is carveable by half-planes that runs in expected time $O((\log n/\log r) \cdot (T(O(n), O(nr \log n)) + nr \log n))$ with high probability.*

Proof. We use a bottom-up sampling approach. Let E_0 be the edges (i.e. line segments not including end points) defining the polytope P. We choose $E_0 \supset E_1 \supset E_2 \supset \cdots \supset E_k = \varnothing$ to be a series of uniformly random samples of the edges: To get E_{i+1} from E_i, we take each edge $e \in E_i$ with probability $\frac{1}{r}$ for some parameter r. We stop at the first value k such that $E_k = \varnothing$. Note that with high probability (w.h.p.) $k = \Theta(\log n/\log r)$, as $|E_0| = O(n)$.

Throughout the algorithm, for each query $q = (v, T)$ pair (with T contained in a plane L_T), we maintain two half-planes $H_i^{\uparrow}(q), H_i^{\downarrow}(q) \subset L_T \setminus \mathrm{CH}(E_i \cap L_T)$. Their boundaries are, respectively, the upper and lower tangents of $\mathrm{CH}(E_i \cap L_T)$ passing through v. Note that it is possible that $\mathrm{CH}(E_i \cap L_T)$ is empty (i.e. when $i = k$), so in this case, we let $H_i^{\uparrow}(q) = H_i^{\downarrow}(q) = L_T$. See Figure 4 for an illustration. Let $\mathcal{H}_i = \{H_i^{\uparrow}(q) \mid q \in \mathcal{Q}\} \cup \{H_i^{\downarrow}(q) \mid q \in \mathcal{Q}\}$.

To compute $\mathcal{H}_i$ from $\mathcal{H}_{i+1}$, we create an instance of intersection detection between the line segments $E_i \setminus E_{i+1}$ and the half-planes $\mathcal{H}_{i+1}$.

We analyze what intersections can occur for the half-planes defined by a query $q = (v, T)$. We observe that on the plane L_T, since E_{i+1} is a $1/r$ sample of E_i, $L_T \cap E_{i+1}$ is also a $1/r$ sample of $L_T \cap E_i$. By a standard analysis of Clarkson and Shor [5], this implies that the number of points of $L_T \cup E_i$ that lie within $H_{i+1}^{\uparrow}(q)$ and $H_{i+1}^{\downarrow}(q)$ is at most $O(r \log n)$ w.h.p. Note that this is exactly the intersections between $E_i \setminus E_{i+1}$ and $H_{i+1}^{\uparrow}(q)$ and $H_{i+1}^{\downarrow}(q)$, and thus the total number of intersections between $E_i \setminus E_{i+1}$ and $\mathcal{H}_{i+1}$ is $O(nr \log n)$.

For a query $q = (v, T)$, three types of events may occur:

1. $E_{i+1} \cap L_T$ **was empty and** $E_i \cap L_T$ **is non-empty.** In this case, we can inspect the points of $E_i \cap L_T$ (of which there at most $O(r \log n)$ w.h.p.) and either compute $H_i^{\uparrow}(q)$ and $H_i^{\downarrow}(q)$ or deduce that P is not carveable in linear time in the size of $E_i \cap L_T$ as in Theorem 2.
2. $E_i \cap L_T$ **contains a point that lies in both** $H_{i+1}^{\uparrow}$ **and** $H_{i+1}^{\downarrow}$**.** Let e denote the edge that induces this point. Let $e^{\uparrow}$ and $e^{\downarrow}$ be the edges of E_{i+1} that defined $H_{i+1}^{\uparrow}$ and $H_{i+1}^{\downarrow}$ respectively. Observe that $e, e^{\uparrow}, e^{\downarrow}$ are the triple of edges that certify that P is not carvable by Observation 2.
3. **The points of** $E_i \cap L_T$ **either lie in** $H_{i+1}^{\uparrow}(q)$ **or** $H_{i+1}^{\downarrow}(q)$**.** In this case, we can compute $H_i^{\uparrow}(q)$ and

$H_i^{\downarrow}(q)$ from $E_i \cap L_T$ in linear time following a similar procedure to find the most extreme halfplane as in Theorem 2.

In all cases, either we conclude that the polytope P is not carveable or we compute $\mathcal{H}_i$. If we have computed $\mathcal{H}_0$, we have extreme half-planes for every vertex of every facial triangle T. At this point, we can conclude that no vertex of any facial triangle T lies within $\mathrm{CH}\left(\mathrm{int}\,(P) \cap L_T\right)$, but it is still possible that some portion of the interior of a facial triangle T intersects $\mathrm{CH}\left(\mathrm{int}\,(P) \cap L_T\right)$. However, as we computed the extreme half-planes for every vertex, we can check in $O(1)$ time whether one of these half-planes is a separating half-plane satisfying Theorem 1(c).

To analyze the runtime, we use an algorithm for intersection detection between half-planes and line segments $O(\log n / \log r)$ times with $O(n)$ half-planes and $O(nr \log n)$ intersections. Step 1 and 3 of the above run in time linear in the number of intersections, i.e. $O(nr \log n)$ total time. □

Since line segment intersection can be reduced to ray shooting among halfplanes, the algorithm of Agarwal and Matoušek[1] for ray shooting imply that $T(n,k) = O(n^{3/2+\varepsilon} + n^{1/2+\varepsilon} \cdot k)$ for any $\varepsilon > 0$. Thus we conclude the following corollary by choosing $r = O(1)$.

Corollary 4 *For any $\varepsilon > 0$, there exists a Las Vegas algorithm to determine if a polytope P is carveable by half-planes that runs in time $O(n^{3/2+\varepsilon})$ with high probability. Furthermore, if P is carveable this algorithm outputs a set of cuts to carve P.*

We note that using the intersection reporting data structure between triangles[1] and line segments in $\mathbb{R}^3$ by Ezra and Sharir [11] gives a better runtime of $T(n,k) = O(n^{3/2+\varepsilon} + k \log n)$, but does not improve the overall runtime of our algorithm. It is plausible to believe that this exponent of 3/2 is the best we can hope for due to lower bounds for Hopcroft's problem in 3D [10].

3 Ray Sweeps

In this section, we consider cutting material with rays. This models cuts that can be performed with various kinds of tools, such as a powerful waterjet [2], or a laser cutter [3]. Given a target polytope P, we wish to devise a set of cuts to carve P out of arbitrary initial material $C \supset P$. In particular, C is cut into pieces, and one of the pieces is $\mathrm{int}\,(P)$ (and the remaining are leftovers which can be discarded). However, we would like this to be independent of C, so we may use the same set of cuts to carve P from different pieces of initial material. We also classify shapes P that admit such a carving.

[1] A half-plane can be simulated by a sufficiently large triangle.
[2] `https://youtu.be/pemgwRrCs78`
[3] `https://youtu.be/J2oyk3ck8Z8`

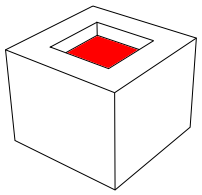


Figure 6: A polytope for which all faces are externally visible that cannot be cut with ray sweeps, since ray sweeps require bounded length. In particular, the shaded face would require a ray sweep of infinite length (i.e., a space-filling curve).

3.1 Model

We would like our model to capture the finite set of cuts that can be made using the tools described previously, and they are commonly operated by moving in a sweeping motion to separate material. Thus, we define our cuts as a set of *ray sweeps*. Consider a ray R defined by an endpoint a and interior point b. A sweep is an interpolation where a or b (or both) travel along an arbitrary continuous path of bounded length.

The reason we require bounded length is because without it, our model would allow us to cut entire faces using the endpoint of a ray, via space-filling curves. Figure 6 is an example of a polytope which *cannot* be cut using ray sweeps, specifically because of the bounded length requirement. Without this requirement, any polytope for which all faces are entirely externally visible could be carved. We use this more restrictive model because it is more representative of how an actual machine could be used to get flat faces.

Given a (triangulated) polytope P, we determine if there is a set of ray sweeps $\mathcal{R}$ such that, for any P-containing set $C \supset P$, the set $C \setminus (\cap_{R \in \mathcal{R}} R)$ has a connected component (i.e., maximal connected open subset) equal to $\mathrm{int}\,(P)$. If such a set of ray sweeps exists, then we also will be able to output it.

As in subsection 2.2, since we only allow finitely many sweeps of bounded length (and hence do not permit space-filling curves), it suffices to ask if each facial triangle can be cut independently. Thus, in this section, we attempt to solve the following two-dimensional problem: Given a triangle $T \subset \mathbb{R}^2$ on the plane, and a set $B_T \subset \mathbb{R}^2$ which is the disjoint union of simple polygons, is there a set of ray sweeps $\mathcal{R}$ inside the plane such that $T \subset \bigcup_{R \in \mathcal{R}} R$ and each $R \in \mathcal{R}$ has $\mathcal{R} \cap \mathrm{int}\,(B_T) = \varnothing$? If we can solve this 2D problem in time $t(k)$, where k is the number of vertices forming the polygons of B_T, then we can classify triangulated polytopes P that can be carved using ray sweeps in time $O(n \cdot t(n))$.

In fact, we show that sweeps of a special form suffice: A *linear ray sweep* is one for which the endpoint a can be linearly interpolated between two points, and the interior point b is constant.

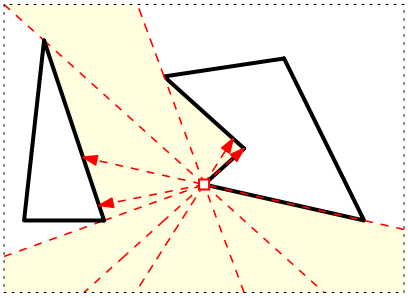


Figure 7: An example of the angle sweep performed in the proof of Theorem 5 around a vertex v. The events are denoted with the red lines/rays, and the valid ray sweeps are coloured.

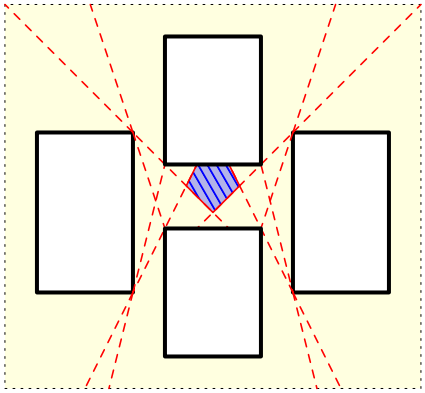


Figure 8: An example of a pentagon that cannot be cut with ray sweeps for a given set of obstacles. The bordering linear ray sweeps are shown.

3.2 2D Cuttable Regions

Compared to Section 2, our approach in this section is reversed. Rather than directly checking if each triangle can be carved, we first map out the carveable regions along the plane containing the triangle, and then afterwards we check if the triangle itself is contained within those regions.

Theorem 5 *Given a set $B \subset \mathbb{R}^2$ which is the disjoint union of simple polygons with a total of k vertices, there is an $O(k^2 \log k)$ time algorithm that can find a set of linear ray sweeps with total complexity $O(k^2)$, the union of which is exactly the union of all rays $R \subset \mathcal{R}$ such that $R \cap \text{int}(B) = \varnothing$.*

Proof. For each vertex v of B, we compute a set of linear ray sweeps A_v, each of which passes through v, and none of which intersect $\text{int}(B)$. At a high level, this algorithm rotates a line passing through v, and maintains the maximal rays in both directions through the line (if any) that include v and do not intersect $\text{int}(B)$.

For the vertex v in B, in order to compute A_v, we perform an angle sweep around v, where we rotate a line that at all times passes through v. The events of our angle sweep are the set of other vertices in B. Compute the ray starting from v through every other vertex in B, and sort all other vertices in B by the angles of those rays (with an arbitrary branch cut). The other vertices form the events of our angle sweep. Now, perform an angle sweep that continuously sweeps a line l, while maintaining a data structure containing the order of all edge interiors in B (i.e., edges without their endpoints) intersected crossed by l, along with their order relative to v along l. Any standard binary search tree suffices for this purpose. Note that the wording *crossed* implies that it is okay for l to intersect an edge interior to which it is parallel. Between two events, we check if along l there is a ray from v to infinity that does not intersect any edge interiors in B (i.e., we check if v is either the first or last element along l according to our data structure). If so, we have found a linear ray sweep between these two events. Extend the ray backwards to the first/last element along l (depending on the direction) at both events. The linear ray sweep then linearly interpolates between those two extended rays, using v as a pivot point. See Figure 7 for an illustration.

To show the correctness of this algorithm, first observe that any ray passing through a vertex of B is contained in one of the linear ray sweeps. Then, we claim that any valid ray V^* not intersecting a vertex of B can be cut using rays which intersect at least one vertex of B. To see this, let $V \subseteq V^*$ be a ray with endpoint $p \in V^*$, an arbitrary point cut by V^*. Consider rotating V around p until it hits a vertex (possibly many) of B, call this new ray V'. Observe that V' intersects a vertex of B as desired, and still contains p. Since p was an arbitrary point of V^*, the claim follows. Hence, our algorithm produces linear ray sweeps that together cut all rays not intersecting B.

Since there are k vertices and each angle sweep takes $O(k \log k)$ time both to sort and to perform, this algorithm runs in $O(k^2 \log k)$ time in total. □

3.3 Decision Algorithm

Theorem 6 *Let B be a set of interior-disjoint simple polygons in $\mathbb{R}^2$ with a total of k vertices, let T be a triangle in $\mathbb{R}^2$. Then, in $O(k^4)$ time, we can check if there is a set of ray sweeps $\mathcal{R}$ such that $T \subset \cup_{R \in \mathcal{R}} R$ so that each ray sweep $R \in \mathcal{R}$ has $R \cap \text{int}(B) = \varnothing$. If there is, then we can also output it in the same time complexity.*

Proof. Apply Theorem 5 to get a set of $O(k^2)$ linear ray sweeps whose union is equal to the set of all area that can be carved via rays that do not intersect $\text{int}(B)$. Take the arrangement of all lines containing each ray or line segment forming the boundary of each

ray sweep. This arrangement can be computed as a doubly-connected edge list in $O(k^4)$ time using a standard incremental construction [7, Theorem 8.6].

For each edge in the graph formed by the arrangement, we include information about which ray sweep boundaries contain this edge. Then, we can traverse the cells of the arrangement with a depth-first search, while maintaining the current set of regions in which the current cell is contained, updating as we traverse each edge. We record, for each cell, whether or not it is part of any region. In this way, we obtain information about which cells are included in the union of the regions in $O(k^4)$ time (linear in the complexity of the arrangement).

Finally, consider each cell of the arrangement. Check if the cell intersects T. If so, check if it is marked as being in the interior of at least one linear ray sweep that can be carved. If it is not, then T cannot be carved. If we determine that all cells intersecting T can be carved, then T itself can also be carved. The time complexity to check if each cell intersects T is also $O(k^4)$, and hence the total time complexity of this algorithm is $O(k^4)$, as desired. □

Corollary 7 *Let P be a triangulated polytope with n faces. Then, there is an algorithm running in $O(n^5)$ time which can determine if P can be carved with ray sweeps. Moreover, if it can, then the algorithm can output a set of linear ray sweeps carving P.*

Proof. Consider each of the facial triangle T of P separately. Let L_T be the plane containing T, and compute the set of disjoint open polygonal regions $B_T = L_T \cap \operatorname{int}(P)$. If B_T is empty, then T can be cut with a single ray sweep. If B_T is non-empty, apply Theorem 6 to determine if T can be cut with ray sweeps. Maintain a list of ray sweeps used, to be output if all triangles can be cut with ray sweeps. There are n facial triangles, hence this algorithm runs in $O(n^5)$ time. □

4 Conclusion

In this paper we discussed two models (half-planes and ray sweeps) of carving three-dimensional polytopes. We focused on the decision variant of each, while retaining the ability to generate a list of cuts when the input polytopes are carveable. Interestingly, even when a polytope P is not carveable our algorithms can all be modified to find a minimal carveable polytope that contains P. This could be quite useful in real-world applications, where a small number of additional cuts could be made using a more specialized tool.

There are several natural resulting open questions from our work:

- Is there a deterministic algorithm for half-plane carving running in subquadratic time?
- Is there a faster algorithm for 2D ray sweep carving that makes use of the triangles to be carved directly?
- Are there efficient algorithms for optimization variants of our problems? Either minimizing total length of cuts or minimizing the number of cuts.

References

[1] P. K. Agarwal and J. Matousek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.

[2] S. I. Ahmed, M. Hasan, and M. A. Islam. Cutting a convex polyhedron out of a sphere. *Graphs Comb.*, 27(3):307–319, 2011.

[3] S. Bereg, O. Daescu, and M. Jiang. A PTAS for cutting out polygons with lines. *Algorithmica*, 53(2):157–171, 2009.

[4] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[5] K. L. Clarkson and P. W. Shor. Application of random sampling in computational geometry, II. *Discret. Comput. Geom.*, 4:387–421, 1989.

[6] O. Daescu and J. Luo. Cutting out polygons with lines and rays. *Int. J. Comput. Geom. Appl.*, 16(2-3):227–248, 2006.

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

[8] E. D. Demaine, M. L. Demaine, and C. S. Kaplan. Polygons cuttable by a circular saw. *Comput. Geom.*, 20(1-2):69–84, 2001.

[9] A. Dumitrescu and M. Hasan. Cutting out polygons with a circular saw. *Int. J. Comput. Geom. Appl.*, 23(2):127–140, 2013.

[10] J. Erickson. New lower bounds for hopcroft's problem. *Discret. Comput. Geom.*, 16(4):389–418, 1996.

[11] E. Ezra and M. Sharir. On ray shooting for triangles in 3-space and related problems. In *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference)*, volume 189 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[12] J. W. Jaromczyk and M. Kowaluk. Sets of lines and cutting out polyhedral objects. *Comput. Geom.*, 25(1-2):67–95, 2003.

[13] M. H. Overmars and E. Welzl. The complexity of cutting paper (extended abstract). In *Proceedings of the First Annual Symposium on Computational Geometry, Baltimore, Maryland, USA, June 5-7, 1985*, pages 316–321. ACM, 1985.

[14] X. Tan. Approximation algorithms for cutting out polygons with lines and rays. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 534–543. Springer, 2005.

Improved upper bounds for the Heilbronn's Problem for k-gons

Rishikesh Gajjala* Jayanth Ravi

Abstract

The Heilbronn triangle problem asks for the placement of n points in a unit square that maximizes the smallest area of a triangle formed by any three of those points. In 1972, Schmidt considered a natural generalization of this problem. He asked for the placement of n points in a unit square that maximizes the smallest area of the convex hull formed by any four of those points. He showed a lower bound of $\Omega(n^{-3/2})$, which was improved to $\Omega(n^{-3/2}\log n)$ by Leffman.

A trivial upper bound of $3/n$ could be obtained and Schmidt asked if this can be improved asymptotically. However, despite several efforts, no asymptotic improvement over the trivial upper bound was known for the last 50 years, and the problem started to get the tag of being notoriously hard. Szemerédi posed the question of whether one can, at least, improve the constant in this trivial upper bound. In this work, we answer this question by proving an upper bound of $2/n + o(1/n)$. We also extend our results to any convex hulls formed by $k \geq 4$ points.

1 Introduction

Given a constant $k \geq 3$ and a set $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ of $n \geq k$ points on the unit square $[0,1]^2$, let $\mathcal{A}_k(P)$ be the area of the smallest convex hull among all convex hulls determined by subsets of k points in $\mathcal{P}$. The supremum value of $\mathcal{A}_k(P)$ over all choices of $\mathcal{P}$ is denoted by $\Delta_k(n)$. The Heilbronn triangle problem asks for the value of $\Delta_3(n)$.

The Heilbronn triangle problem is one of the fundamental problems in discrete geometry and discrepancy theory and has a rich history. Paul Erdős proved that $\Delta_3(n) = \Omega\left(\frac{1}{n^2}\right)$. This was believed to be the upper bound for some time until Komlós, Pintz and Szemerédi [12] proved that

$$\Delta_3(n) = \Omega\left(\frac{\log n}{n^2}\right)$$

Over a series of works, the upper bounds were improved by Roth [16, 17, 18, 19] and Schmidt [20]. The current best-known upper bound is due to Komlós, Pintz and Szemerédi [11]

$$\Delta_3(n) = \mathcal{O}\left(\frac{2^{c\sqrt{\log n}}}{n^{8/7}}\right)$$

This has been recently claimed to be improved by Cohen, Pohoata and Zakharov [7] to $\mathcal{O}\left(n^{-8/7-1/2000}\right)$.

There has also been work on several variants of this problem. Jiang, Li and Vitany [10] and Benevides, Hoppen, Lefmann and Odermann [4] studied the case in which the points were randomly distributed. The problem was also explored in higher dimensions by placing n points in d-dimensional unit cubes $[0,1]^d$ instead of a unit square [1, 2, 3, 6, 13, 15].

Schmidt asked about the value of $\Delta_k(n)$ and proved that $\Delta_4(n) = \Omega\left(\frac{1}{n^{1.5}}\right)$ [20]. Bertraln-Kretzberg, Hofmeister and Lefmann generalized this result to k-gons by proving that $\Delta_k(n) = \Omega\left(\frac{1}{n^{\frac{k-1}{k-2}}}\right)$ [5]. This was improved by Lefmann[14] to

$$\Delta_k(n) = \Omega\left(\frac{(\log n)^{1/k-2}}{n^{1+\frac{1}{k-2}}}\right)$$

2 Our results

A trivial upper bound of $\Delta_4(n) \leq \frac{3}{n}$ can be obtained by subdividing the unit square into squares of side length $\sqrt{\frac{3}{n}}$ using the pigeonhole principle. However, despite several efforts to improve this upper bound (asymptotically) since it was posed in 1972, there has been no progress! Szemerédi asked if at least the constants in this upper bound can be improved [21]. In this work, we answer this question by proving the following theorem.

Theorem 1 $\Delta_4(n) \leq \frac{2}{n} + o\left(\frac{1}{n}\right)$

We also generalize our result to general k-gons for any constant $k \geq 4$.

Theorem 2 $\Delta_k(n) \leq \frac{k-2}{n} + o\left(\frac{1}{n}\right)$

*Indian Institute of Science, Bengaluru, rishikeshg@iisc.ac.in

3 Convex quadrilaterals: Proof of Theorem 1

We solve a more general problem by having the points on a unit rectangle (instead of a unit square). Given a set $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ of $n \geq 3$ points on the unit rectangle $[0, d] \times [0, d^{-1}]$ and $k \leq n$, let $\mathcal{A}'_k(P)$ be the minimum area of the convex hull determined by a set of k points in $\mathcal{P}$ for any $0 < d \leq 1$. The supremum value of $\mathcal{A}'_k(P)$ over all choices of $\mathcal{P}$ is denoted by $\Delta'_k(n)$. It is easy to see that by definition

$$\Delta_k(n) \leq \Delta'_k(n)$$

For $k = 4$, when there are n points, in the argument to obtain a trivial bound of $\Delta'_4(n) \leq \frac{3}{n-1}$, we partition the unit rectangle into at most $(n-1)/3$ smaller rectangles. This would guarantee that there exists a small rectangle containing at least 4 points. Naturally, one can also extend this idea to make sure there are at most $\left(\frac{n-1}{n'-1}\right)$ smaller rectangles and force $n' \geq 4$ points into one rectangle. This gives us a relation between $\Delta'_4(n)$ and $\Delta'_4(n')$, which is formalized in Observation 3.

Observation 3 *For $4 \leq n' \leq n$,*

$$\Delta'_4(n) \leq \left\lfloor \frac{n-1}{n'-1} \right\rfloor^{-1} \Delta'_4(n')$$

Proof. Partition the unit area into a grid with $\left\lfloor \frac{n-1}{n'-1} \right\rfloor$ rectangles of area $\left\lfloor \frac{n-1}{n'-1} \right\rfloor^{-1}$. Since there are n points and $\left\lfloor \frac{n-1}{n'-1} \right\rfloor \leq \frac{n-1}{n'-1}$ rectangles, by the pigeonhole principle, one of the smaller rectangles (with their boundary included) has at least n' points. Therefore, there always exists n' points within an area at most $\left\lfloor \frac{n-1}{n'-1} \right\rfloor^{-1}$. It now follows by a scaling argument that there exist four points within an area at most $\left\lfloor \frac{n-1}{n'-1} \right\rfloor^{-1} \Delta'_4(n')$. □

When $n' = 4$, this gives the trivial bound of $\Delta'_4(n) \leq \frac{3}{n-3} \approx 3/n$ as expected. We can do slightly better by tuning the value of n' to be 6. We start with finding the exact value of $\Delta'_4(6)$.

Observation 4 $\Delta'_4(6) = 1/2$

Proof. Let C be the centre of the rectangle $[0, d] \times [0, d^{-1}]$ and $\mathcal{P}$ be a set of any six points in $[0, d] \times [0, d^{-1}]$. Pick an arbitrary point $P \in \mathcal{P}$ and extend the line segment PC into a line. The extended line PC cuts the rectangle $[0, d] \times [0, d^{-1}]$ into two convex parts, and by symmetry, both these parts have the same area, i.e., $1/2$. From the pigeonhole principle, at least 3 of the remaining 5 points lie on one side of the extended line PC. Therefore, 4 points (including P) exist, which are contained in a convex shape whose area is $1/2$. Therefore, $\Delta'_4(6) \leq 1/2$.

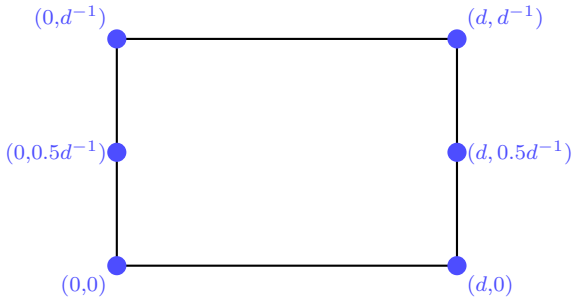


Figure 1: $\Delta'_4(6) \geq 1/2$

It may be noted the bound of $1/2$ is achieved in Figure 1. Therefore $\Delta'_4(6) \geq 1/2$ □

Corollary 5 $\Delta_4(n) \leq \frac{1}{2}\left\lfloor \frac{n-1}{5} \right\rfloor^{-1} \leq \frac{2.5}{n-5}$

Proof. By substituting $n' = 6$ in Observation 3 and using Observation 4, we get

$$\Delta'_4(n) \leq \left\lfloor \frac{n-1}{5} \right\rfloor^{-1} \Delta'_4(6) = \frac{1}{2}\left\lfloor \frac{n-1}{5} \right\rfloor^{-1} \leq \frac{2.5}{n-5}$$

□

We will now extend the idea of Observation 4 to all n of the form $2^s + 2$ for any $s \geq 2$, i.e., for all n of such form, we prove $\Delta_4(n) \leq \frac{2}{n-2}$ in Theorem 8 using Observation 6 and Lemma 7.

Observation 6 *For any point P in a convex polygon C, there exists a line through P which partitions C into two halves of equal area.*

Proof. Let A be the area of C. Pick any arbitrary line L through P and let the areas of the convex polygons on both of its sides of L be L_1 and L_2 such that $L_1 \leq A/2 \leq L_2$. By rotating the line by 180° around P, we get $L_2 \leq 1/2 \leq L_1$. Since L_1 changes continuously as a function of the angle of rotation θ, by the intermediate value theorem, it must have achieved $1/2$ in between for some θ. □

Lemma 7 *If a convex polygon of area Δ has $2^i \cdot \beta + 2$ points, then there is a convex polygon of area $\Delta/2$ which contains at least $2^{i-1} \cdot \beta + 2$ points.*

Proof. Pick one of the $2^i \cdot \beta + 2$ points arbitrarily, say P. From Observation 6, there is a line L through P cutting the polygon into two halves. Note that by the pigeonhole principle, one of the halves would have at least $2^{i-1} \cdot \beta + 1$ points. By including P in this region, we get a convex polygon of area $\Delta/2$, containing at least $2^{i-1} \cdot \beta + 2$ points. □

We first introduce some new notation. Given a set $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ of $n \geq 3$ points on an *arbitrary convex object* $\mathcal{C}$ *of unit area*, let $\mathcal{A}_k^{\mathcal{C}}(Q)$ be the minimum area of the convex hull determined by some k points in $\mathcal{Q}$. The supremum value of $\mathcal{A}_k^{\mathcal{C}}(Q)$ over all choices of $\mathcal{Q}$ is denoted by $\Delta_k^{\mathcal{C}}(n)$. Let $\Delta_k''(n)$ denote the supremum value of $\Delta_k^{\mathcal{C}}(n)$ over all convex objects of unit area. It is easy to see that by definition

$$\Delta_k(n) \leq \Delta_k'(n) \leq \Delta_k''(n)$$

Theorem 8 *If $n = 2^s + 2$ for some integer $s > 0$, then*

$$\Delta_4(n) \leq \Delta_4'(n) \leq \frac{1}{2^{s-1}} = \frac{2}{n-2}$$

Proof. Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ be a set of n points. We will prove a stronger statement of

$$\Delta_4''(n) \leq \frac{1}{2^{s-1}}$$

Observation 9 *For every $i \in [0, s-1]$, there exists a convex polygon of area at most $\frac{1}{2^i}$ which has at least $2^{s-i} + 2$ points from $\mathcal{Q}$.*

Proof. We prove this by induction on i. When $i = 0$, the claim is true by definition. Suppose there exists a convex polygon of area at most $\frac{1}{2^i}$ with at least $2^{s-i}+2$ points from $\mathcal{Q}$, then from Lemma 7, for $i < s-1$, there exists a convex polygon of area at most $\frac{1}{2^{i+1}}$ with at least $2^{s-(i+1)} + 2$ points from $\mathcal{Q}$ □

By substituting $i = s - 1$ in Observation 9, Theorem 8 follows. □

One may note that this would give an upper bound of $\approx 2/n$ for many arbitrarily large n of the form $2^s + 2$. However, there are also several arbitrarily large n of the form, say, $2^s + 1$, for which this bound is not useful. We fix this using Observation 3.

Corollary 10 $\Delta_4(n) \leq \frac{2}{n} + o\left(\frac{1}{n}\right)$

Proof. For every n, there exists some i such that

$$2^{i-1} + 2 \leq n < 2^i + 2$$

Let $n' = 2^{\lceil 0.5i \rceil} + 2$. From Theorem 8,

$$\Delta_4'(n') \leq \frac{2}{n'-2} < \frac{2}{n'-1}$$

$$\left\lfloor \frac{n-1}{n'-1} \right\rfloor > \frac{n-1}{n'-1} - 1 \geq \frac{n-n'}{n'-1}$$

From Observation 3,

$$\Delta_4'(n) \leq \Delta_4'(n') \left\lfloor \frac{n-1}{n'-1} \right\rfloor^{-1} \leq \frac{2}{n'-1} \cdot \frac{n'-1}{n-n'} = \frac{2}{n-n'}$$

$$= \frac{2}{n} + \frac{2n'}{n(n-n')} = \frac{2}{n} + \mathcal{O}\left(\frac{1}{n^{1.5}}\right)$$

□

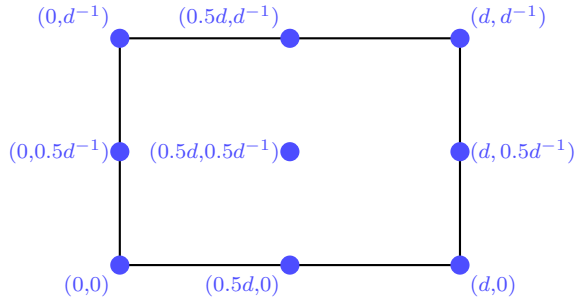


Figure 2: $\Delta_4'(9) \geq 1/4$

From Figure 2, it is easy to see that $\Delta_4'(9) \geq 1/4$. We conjecture that the other direction is also true, i.e., $\Delta_4'(9) \leq 1/4$

Conjecture 1 $\Delta_4'(9) = 1/4$

If true, Conjecture 1 would directly imply that $\Delta_4(n) \leq \frac{2}{n-8}$ from Observation 3 by picking n' to be 9. We also note that finding the exact values of $\Delta_k(n)$ is of independent interest and has been well studied for $k = 3$ [8, 9, 22].

One may note that our analysis will extend to general convex figures with unit area (instead of just unit squares), i.e.,

$$\Delta_4''(n) \leq \frac{2}{n} + o\left(\frac{1}{n}\right)$$

4 Convex k-gons: Proof of Theorem 2

In this section, we extend our proofs for $k = 4$ to give upper bounds on $\Delta_k(n)$ for any constant k.

Proposition 11 (Analogue of Observation 3) *For $n' \leq n$,*

$$\Delta_k'(n) \leq \left\lfloor \frac{n-1}{n'-1} \right\rfloor^{-1} \Delta_k'(n')$$

Proof. The proof of Proposition 11 directly extends from Observation 3. □

Theorem 12 (Analogue of Theorem 8) *If* $n = 2^s(k) - 2^{s+1} + 2$ *for some integer* $s > 0$*, then*

$$\Delta_k(n) \leq \Delta'_k(n) \leq \frac{1}{2^s} = \frac{k-2}{n-2}$$

Proof. Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ be a set of n points. We will prove a stronger statement of

$$\Delta''_k(n) \leq \frac{1}{2^s}$$

Observation 13 *For every* $i \in [0, s]$*, there exists a convex polygon of area at most* $\frac{1}{2^i}$ *with at least* $2^{s-i}(k) - 2^{s+1-i} + 2$ *points from* $\mathcal{Q}$*.*

Proof. We prove this by induction on i. When $i = 0$, the claim is true by definition. Suppose there exists a convex polygon of area at most $\frac{1}{2^i}$ with at least $2^{s-i}(k) - 2^{s+1-i} + 2$ points from $\mathcal{Q}$, then from Lemma 7, for $i < s$, there exists a convex polygon of area at most $\frac{1}{2^{i+1}}$ with at least $2^{s-(i+1)}(k) - 2^{s+1-(i+1)} + 2$ points from $\mathcal{Q}$. □

By substituting $i = s$ in Observation 13, Theorem 12 follows. □

Corollary 14 $\Delta_k(n) \leq \frac{k-2}{n} + o\left(\frac{1}{n}\right)$

Proof. For every n, there exists some i such that

$$2^{i-1}(k) - 2^i + 2 \leq n < 2^i(k) - 2^{i+1} + 2$$

Let $n' = 2^{\lceil 0.5i \rceil}(k) - 2^{\lceil 0.5i \rceil + 1} + 2$. From Theorem 12,

$$\Delta'_k(n') \leq \frac{k-2}{n'-2} < \frac{k-2}{n'-1}$$

$$\left\lfloor \frac{n-1}{n'-1} \right\rfloor > \frac{n-1}{n'-1} - 1 \geq \frac{n-n'}{n'-1}$$

From Proposition 11,

$$\Delta'_k(n) \leq \Delta'_k(n') \left\lfloor \frac{n-1}{n'-1} \right\rfloor^{-1} \leq \frac{k-2}{n'-1} \cdot \frac{n'-1}{n-n'} = \frac{k-2}{n-n'}$$

$$= (\frac{1}{n} + \frac{n'}{n(n-n')}) \cdot (k-2) = \frac{k-2}{n} + \mathcal{O}\left(\frac{1}{n^{1.5}}\right)$$

□

Conjecture 2 $\Delta'_k(\alpha(k-1)) = \frac{1}{2(\alpha-1)}$ *for* $\alpha \leq k-1$

One can prove that $\Delta'_k(\alpha(k-1)) \geq \frac{1}{2(\alpha-1)}$ by placing the points in the corners of a $\alpha \times (k-1)$ grid. We conjecture that this is, in fact, the optimal placement. When $\alpha = 2$, this is indeed true by Lemma 7. Conjecture 1 is a special case of Conjecture 2 when $k = 4$. If true, Conjecture 2 would imply an upper bound of $\approx k/(2n)$

One may notice that Conjecture 2 can not be extended to $\alpha \leq k-1$, as at least k points become collinear in that case.

Acknowledgements

RG thanks Endre Szemerédi for introducing this problem to him during the 9th Heidelberg Laureate Forum. RG thanks Saladi Rahul for his comments on the manuscript.

References

[1] G. Barequet. A lower bound for heilbronn's triangle problem in d dimensions. *SIAM J. Discret. Math.*, 14(2):230–236, 2001.

[2] G. Barequet. The on-line heilbronn's triangle problem. *Discret. Math.*, 283(1-3):7–14, 2004.

[3] G. Barequet and A. Shaikhet. The on-line heilbronn's triangle problem in d dimensions. *Discret. Comput. Geom.*, 38(1):51–60, 2007.

[4] F. S. Benevides, C. Hoppen, H. Lefmann, and K. Odermann. Heilbronn triangle-type problems in the unit square [0, 1] 2. *Random Structures & Algorithms*, 62(3):585–599, 2023.

[5] C. Bertram-Kretzberg, T. Hofmeister, and H. Lefmann. An algorithm for heilbronn's problem. *SIAM J. Comput.*, 30(2):383–390, 2000.

[6] P. Braß. An upper bound for the d-dimensional analogue of heilbronn's triangle problem. *SIAM J. Discret. Math.*, 19(1):192–195, 2005.

[7] A. Cohen, C. Pohoata, and D. Zakharov. A new upper bound for the heilbronn triangle problem. *arXiv preprint arXiv:2305.18253*, 2023.

[8] F. Comellas and J. L. A. Yebra. New lower bounds for heilbronn numbers. *the electronic journal of combinatorics*, pages R6–R6, 2002.

[9] L. Dehbi and Z. Zeng. Heilbronn's problem of eight points in the square. *Journal of Systems Science and Complexity*, 35(6):2452–2480, 2022.

[10] T. Jiang, M. Li, and P. M. B. Vitányi. The average-case area of heilbronn-type triangles. *Random Struct. Algorithms*, 20(2):206–219, 2002.

[11] J. Komlós, J. Pintz, and E. Szemerédi. On heilbronn's triangle problem. *Journal of the London Mathematical Society*, 2(3):385–396, 1981.

[12] J. Komlós, J. Pintz, and E. Szemerédi. A lower bound for heilbronn's problem. *Journal of the London Mathematical Society*, 2(1):13–24, 1982.

[13] H. Lefmann. On heilbronn's problem in higher dimension. *Comb.*, 23(4):669–680, 2003.

[14] H. Lefmann. Distributions of points in the unit square and large k-gons. *Eur. J. Comb.*, 29(4):946–965, 2008.

[15] H. Lefmann and N. Schmitt. A deterministic polynomial-time algorithm for heilbronn's problem in three dimensions. *SIAM J. Comput.*, 31(6):1926–1947, 2002.

[16] K. Roth. On a problem of heilbronn, ii. *Proceedings of the London Mathematical Society*, 3(2):193–212, 1972.

[17] K. Roth. Estimation of the area of the smallest triangle obtained by selecting three out of n points in a disc of unit area. In *Proc. of Symposia in Pure Mathematics*, volume 24, pages 251–262. AMS Providence, 1973.

[18] K. Roth. Developments in heilbronn's triangle problem. *Advances in Mathematics*, 22(3):364–385, 1976.

[19] K. F. Roth. On a problem of heilbronn. *Journal of the London Mathematical Society*, 1(3):198–204, 1951.

[20] W. M. Schmidt. On a Problem of Heilbronn†. *Journal of the London Mathematical Society*, s2-4(3):545–550, 04 1972.

[21] E. Szemerédi. Personal communication, September, 2022.

[22] Z. Zeng and L. Chen. On the heilbronn optimal configuration of seven points in the square. In *International Workshop on Automated Deduction in Geometry*, pages 196–224. Springer, 2008.

The exact balanced upper chromatic number of the n-cube over t elements *

Gabriela Araujo-Pardo† Silvia Fernández-Merchant‡ Adriana Hansberg§ Dolores Lara¶
Amanda Montejano‖ Déborah Oliveros**

Abstract

We consider colorings of the cube C_t^n defined as the set of lattice points in $[0, t-1]^n$. The *geometric lines* of this cube are all the subsets of t collinear points; these are the lines typically used in multidimensional tic-tac-toe. Given a coloring, a geometric line is *rainbow* if all its points have different colors. The coloring is *balanced* if the color class sizes differ in at most one. In this paper, we determine the exact value of the *balanced upper chromatic number* of C_t^n, for any positive integers n and $t \geq 4n-2$. That is, we find the largest integer k for which there is a balanced k-coloring of C_t^n without rainbow geometric lines. This problem is related to the impossibility of the existence of a rainbow Ramsey counterpart of the famous Hales–Jewett theorem in Ramsey theory.

1 Introduction

The *n-cube on t elements*, denoted by C_t^n, is the set of lattice points with integer coordinates in the interval $[0, t-1]$. The *geometric lines* of this cube are all the subsets of t collinear points. They satisfy that, for each $0 \leq i \leq t-1$, the i^{th} coordinates of the t points are either all equal or all different. These correspond to the lines typically used in multidimensional tic-tac-toe games [6].

We consider colorings of the points of C_t^n. Given a coloring, we say that a geometric line is *rainbow* if its t points are colored with t different colors. A coloring is *balanced* if each color is used almost the same number of times, more precisely, if all color class sizes differ by at most one. (When the color classes are exactly the same size, these colorings are also called *equinumerous* [19].) A coloring is *rainbow-free* if none of its geometric lines are rainbow. The largest integer k for which there is a balanced rainbow-free k-coloring of C_t^n is called the *balanced upper chromatic number* of C_t^n. In this paper, we determine the balanced upper chromatic number of C_t^n, for every $n \geq 2$ and every $t \geq 4n-2$.

To place this problem in a general context, we look back in history. Starting in the 1930s, Paul Erdős and George Szekeres [13] popularized what is now known as *Ramsey theory* [15]. This theory is based on Ramsey's idea that any sufficiently large structure contains a *regular* substructure [24] or, in the words of Theodore Motzkin, "complete disorder is impossible" [23]. An interesting branch of this area is what is known in the literature as *rainbow* or *anti Ramsey theory* (see [9, 10, 12] and many other references cited in [19]), which studies the existence of special rainbow (totally multicolored) subsets, provided that the color ground set is sufficiently large and that all colors are well represented. In contrast, Ramsey theory (in the context of colorings) studies the existence of special monochromatic (one color) subsets in colorings of large enough sets.

Many results in Ramsey theory have a rainbow Ramsey version. For example, van der Waerden's theorem [25, 26] states that, for n large enough, any k-coloring of the integers $\{1, 2, 3, \ldots, n\}$ contains a monochromatic arithmetic progression of a given length t. The rainbow Ramsey versions of this result usually impose the condition that each color appears with a minimum density. The equinumerous version for 3-term arithmetic progressions and 3 colors was proved in [20]. This result was originally conjectured in [19], where they explore other rainbow variants. In particular, they propose this interesting problem: for fixed $t \geq 3$, find the minimum number of colors k such that, for every n, any balanced k-coloring of the set $\{1, 2, \ldots, kn\}$ has a rainbow arithmetic progression of length t. This is similar to the line of work of this paper (see more below).

A deviation of this trend occurs when looking at the

*Part of this research was performed during a Simons Laufer Mathematical Sciences Institute (SLMath, formerly MSRI) summer research program, which is supported by the National Science Foundation (Grant No. DMS-1928930) and in partnership with the Mathematics Institute of the National Autonomous University of Mexico (UNAM).

†Instituto de Matemáticas, Universidad Nacional Autónoma de México, Campus Juriquilla, Mexico, garaujo@im.unam.mx, supported by DGAPA PAPIIT IN113324

‡California State University, Northridge, 18311 Nordhoff St, Northridge, CA, 91330, USA. silvia.fernandez@csun.edu, supported by a 2024 RSP CSUN Campus Funding Initiative

§Instituto de Matemáticas, Universidad Nacional Autónoma de México, Campus Juriquilla, Mexico, ahansberg@im.unam.mx, supported by DGAPA PAPIIT IG100822

¶Departamento de Computación, Cinvestav, Mexico City, Mexico, dolores.lara@cinvestav.mx

‖UMDI, Facultad de Ciencias, UNAM Juriquilla, Querétaro, Mexico, amandamontejano@ciencias.unam.mx, supported by DGAPA PAPIIT IG100822

**Instituto de Matemáticas, Universidad Nacional Autónoma de México, Campus Juriquilla, Mexico, doliveros@im.unam.mx, supported by DGAPA PAPIIT 35-IN112124

famous Hales-Jewett theorem [6, 16] (a generalization of van der Waerden's theorem) that establishes the existence of monochromatic geometric lines for any k-coloring of the cube C_t^n and n large enough. (This theorem is actually stronger guaranteeing the existence of a monochromatic *combinatorial line*. The set of combinatorial lines is a subset of the set of geometric lines as defined above, see Section 1.1 or [6] for a precise definition.) The rainbow Ramsey version of this result would state that, given k and t ($k \geq t$), any k-coloring of C_t^n contains a rainbow geometric line, for any n suficiently large. However, this has been disproved even for balanced colorings [19, 22] (see the note after Inequality (2) in Section 1.2). Given this impossibility, a natural direction is to investigate how large the number of colors k needs to be in order to guarantee a rainbow geometric line in any balanced k-coloring of C_t^n. This threshold is settled in this paper for any n and $t \geq 4n-2$. (Note that as long as the number of colors in a balanced coloring is larger than the balanced upper chromatic number of C_t^n, the existence of a rainbow geometric line is guaranteed.)

The balanced upper chromatic number has been studied for cyclic projective planes, projective spaces, for the desarguesian projective planes, and for the cube [4, 8, 22]. A similar parameter has been studied in the setting of hypergraphs for general (not necessarily balanced) colorings [3, 5, 27] and for mixed colorings [21]. Similar approaches involving arithmetic structures are handled in [1, 11, 14, 18].

1.1 The n-cube and its geometric lines

We consider the *n-cube over t elements*, denoted by C_t^n, defined as the set of *points* (ordered n-tuples) on the set $\{0, 1, \ldots, t-1\}$. That is,

$$C_t^n = \{\mathbf{x} = (x_1, x_2, \ldots, x_n) \,:\, 0 \leq x_i \leq t-1, x_i \in \mathbb{Z}\}.$$

We typically use "bold fonts" to represent the points $\mathbf{x}$ of C_t^n as above. A *geometric line* in the n-cube C_t^n consists of exactly t collinear points $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_{t-1}$ of C_t^n. Formally, a set of t distinct points of C_t^n is a geometric line if there is an order of the points, such that when we write their coordinates in the following array

$$\begin{array}{lccccccc}
\mathbf{x}_0 & = (x_{0,1}, & x_{0,2}, & \ldots & x_{0,j}, & \ldots & x_{0,n-1}, & x_{0,n}) \\
\mathbf{x}_1 & = (x_{1,1}, & x_{1,2}, & \ldots & x_{1,j}, & \ldots & x_{1,n-1}, & x_{1,n}) \\
\mathbf{x}_2 & = (x_{2,1}, & x_{2,2}, & \ldots & x_{2,j}, & \ldots & x_{2,n-1}, & x_{2,n}) \\
\vdots & & & & \vdots & & & \vdots \\
\mathbf{x}_{t-1} & = (x_{t-1,1}, & x_{t-1,2}, & \ldots & x_{t-1,j}, & \ldots & x_{t-1,n-1}, & x_{t-1,n})
\end{array}$$

each of the n columns satisfies one of the following conditions:

(a) The entries are all equal to some fixed value $a \in \{0, 1, \ldots, t-1\}$.

(b) The entries appear in increasing order $0, 1, \ldots, t-1$.

(c) The entries appear in decreasing order $t-1, t-2, \ldots, 1, 0$.

(The lines satisfying only conditions (a) and (b) are called *combinatorial lines*). Hence, each line in C_t^n can be identified with a vector $\langle l_1, l_2, \ldots, l_n \rangle$, where the entries correspond to the types, (a), (b) or (c), of each column. That is, for every $1 \leq i \leq n$, $l_i \in \{b, c, 0, 1, 2, \ldots, t-1\}$. Note that columns of type (a) are indicated by the actual fixed value l_i to be used. For example, the line L with a vector $\langle 21, b, 37, 6, b, c, 6, 34, 2, 10 \rangle$ is identified as the following line in C_{41}^{10}.

$$\begin{array}{lcccccccccc}
\mathbf{x}_0 & = (21, & 0, & 37, & 6, & 0, & 40, & 6, & 34, & 2, & 10) \\
\mathbf{x}_1 & = (21, & 1, & 37, & 6, & 1, & 39, & 6, & 34, & 2, & 10) \\
\mathbf{x}_2 & = (21, & 2, & 37, & 6, & 2, & 38, & 6, & 34, & 2, & 10) \\
\vdots & & & & & \vdots & & & & & \vdots \\
\mathbf{x}_{38} & = (21, & 38, & 37, & 6, & 38, & 2, & 6, & 34, & 2, & 10) \\
\mathbf{x}_{39} & = (21, & 39, & 37, & 6, & 39, & 1, & 6, & 34, & 2, & 10) \\
\mathbf{x}_{40} & = (21, & 40, & 37, & 6, & 40, & 0, & 6, & 34, & 2, & 10)
\end{array}$$

It is important to observe that there must be at least one column of type (b) or (c) so that the t points are different. By convention, to avoid describing the same line in two different ways, we always assume that the first column of type (b) or (c) must be of type (b). The set of geometric lines of C_t^n is denoted by $\mathcal{L}(C_t^n)$. Since each geometric line is contained in a straight line in $\mathbb{R}^n$, two distinct points in C_t^n belong to at most one geometric line, and two different geometric lines intersect in at most one point. The number of geometric lines is known to be

$$|\mathcal{L}(C_t^n)| = \frac{(t+2)^n - t^n}{2}. \tag{1}$$

as shown in [6]. Note that this fact can easily be seen using our description of the geometric lines above. There are $(t+2)^n$ vectors $\langle l_1, l_2, \ldots, l_n \rangle$ with entries in $\{b, c, 0, 1, 2, \ldots, t-1\}$ and t^n of them have all entries in $\{0, 1, 2, \ldots, t-1\}$. We divide by 2 because in half of these $(t+2)^n - t^n$ vectors (having at least one entry in $\{b, c\}$) the first entry in $\{b, c\}$ is b and in the other half it is c.

1.2 The problem and main result

We consider the following coloring problem. Let H be a hypergraph whose hyperedges have at least two vertices. The *balanced upper chromatic number* of H, denoted by $\overline{\chi}_b(H)$, is defined as the largest integer k for which there is a balanced k-coloring of the vertices of H without rainbow edges [4].

In the rest of the paper, we associate the n-cube over t elements to the hypergraph whose set of vertices is C_t^n and set of hyperedges is $\mathcal{L}(C_t^n)$. In order to simplify the exposition, we abuse the notation refering to this t-uniform hypergraph simply as C_t^n and to its balanced upper chromatic number as $\overline{\chi}_b(C_t^n)$. Our goal is to determine $\overline{\chi}_b(C_t^n)$ for any positive integer n and t large enough (with respect to n).

As C_1^n is a single point, we consider $t \geq 2$. Observe that any two points in the cube C_2^n are in a line. Then, for any positive integer n, $\overline{\chi}_b(C_2^n) = 1$. In general, if we have fewer colors than points on a line, then no line is rainbow. Hence, $\overline{\chi}_b(C_t^n) \geq t-1$. The nontrivial lower bound

$$\overline{\chi}_b(C_t^n) \geq \left(\frac{t}{2}\right)^n \tag{2}$$

was obtained in [22] for any even $t \geq 4$. Note that this implies that given $k \geq t$, for any sufficiently large n (namely, any n such that $(t/2)^n > k$) there are balanced k-colorings of C_t^n without rainbow lines, showing the impossibility of the rainbow Ramsey version of the Hales-Jewett theorem stated in the introduction.

On the other hand, assigning different colors to all points generates a t^n-coloring in which all lines are rainbow. This gives the trivial upper bound $\overline{\chi}_b(C_t^n) \leq t^n - 1$. To avoid rainbow lines, we need to have at least two points of the same color in each line. In this case, we say that the color *blocks* the line. In particular, any coloring with at most $|\mathcal{L}(C_t^n)| - 1$ color classes of size two and all other classes (if any) of size one would fail to block all the lines. Provided that $t^n \geq 2|\mathcal{L}(C_t^n)|$, the smallest of such colorings has $t^n - |\mathcal{L}(C_t^n)| + 1$ colors and thus

$$\overline{\chi}_b(C_t^n) \leq t^n - |\mathcal{L}(C_t^n)|. \tag{3}$$

This implies the following upper bound.

Proposition 1 *Let n and t be positive integers such that $t \geq \dfrac{2}{\sqrt[n]{2}-1}$. Then*

$$\overline{\chi}_b(C_t^n) \leq \frac{3t^n - (t+2)^n}{2}.$$

Proof. This is a direct application of Identity (1) and Inequality (3), noting that the former holds provided that $t^n \geq 2|\mathcal{L}(C_t^n)| = (t+2)^n - t^n$, which is equivalent to $t \geq \dfrac{2}{\sqrt[n]{2}-1}$. □

Surprisingly, this upper bound is best possible for t large enough as shown by our main result.

Theorem 2 *For integers $n \geq 2$ and $t \geq 4n-2$, the balanced upper chromatic number of C_t^n is*

$$\overline{\chi}_b(C_t^n) = \frac{3t^n - (t+2)^n}{2}.$$

We prove this theorem in Section 2 and finish the paper with a series of remarks and open questions in Section 3.

2 Proof of Theorem 2

In order to prove Theorem 2, we reach the upper bound in Proposition 1 by providing an explicit balanced $(\frac{3t^n-(t+2)^n}{2})$-coloring of C_t^n with no rainbow lines, for every $n \geq 2$ and $t \geq 4n-2$. As detailed in Section 2.4, this is equivalent to proving the existence of what we call a *double-matching* of C_t^n, which could be intuitively described as a "disjoint" selection of two points per line. Hence, one could also refer to a double-matching as a *double-covering*, a *double-transversal*, or a *double-SDR* (a "double" System of Distinct Representatives). To complete this task, we use an auxiliary injective function defined in Section 2.1 and heavily make use of the symmetry of the lines within the n-cube as described in Section 2.2. The double-matching is provided in Section 2.3, which includes the core of the proof.

2.1 An injective function

Let m and k be integers with $0 \leq k \leq \frac{1}{2}(m-1)$. The Hall's Marriage Theorem [17] guarantees the existence of an injective function

$$g_{m,k} : \binom{\{0,1,2,\ldots,m-1\}}{k} \to \binom{\{0,1,2,\ldots,m-1\}}{k+1}$$

such that, for any $S \subset \{0,1,2,\ldots,m-1\}$ with k elements, it holds that $S \subset g_{m,k}(S)$. An explicit such function $g_{m,k}$ was given in [2]. This function, adapted to our setting, can be defined as follows. Given $S = \{s_1, s_2, \ldots, s_k\} \subset \{0,1,2,\ldots,m-1\}$ where $s_1 < s_2 < \cdots < s_k$, let $s_0 = -1$, $r = \min\{s_i - 2i : i \in \{0,1,2,\ldots,k\}\}$, and $\phi(S) = \max\{i \in \{0,1,2,\ldots,k\} : s_i - 2i = r\}$, the largest of the subindices for which $s_i - 2i$ is as small as possible. Then $g_{m,k}(S) = S \cup \{1 + s_{\phi(S)}\}$. Note that when $k = 0$, we have that $S = \emptyset$, $\phi(S) = 0$, and $g_{m,k}(S) = \{0\}$.

2.2 Symmetric pairs and equivalence classes

Let $t \geq 2$ and $0 \leq m \leq t-1$ be integers. Let $\overline{m} = \overline{t-1-m} = \{m, t-1-m\}$. We say that m and

$t-1-m$ are *symmetric* and refer to $\overline{m}$ as a *symmetric pair*. Further, we write $\widehat{m}$ to refer to the smallest element in the pair $\overline{m}$. It is important to note that when t is odd and $m=(t-1)/2$, we have that $m=t-1-m$ and so $\overline{m}$ consists of a single element. For convenience, we still call $\overline{(t-1)/2}$ a symmetric *pair* observing that it is actually not used later in the proof when an actual pair of points is selected.

We partition the set of points C_t^n into equivalence classes. We say that two points $(x_1,x_2,\ldots,x_n)$ and $(y_1,y_2,\ldots,y_n)$ are equivalent if $x_i=y_i$ or $x_i+y_i=t-1$ for all $1\le i\le n$, that is, if each pair of corresponding entries is a symmetric pair. The equivalence class containing the point $\mathbf{x}=(x_1,x_2,\ldots,x_n)$ is denoted by $\overline{\mathbf{x}}=\{(y_1,y_2,\ldots,y_n): y_i\in\overline{x_i} \text{ for all } 1\le i\le n\}$.

Note that two points in the same line are equivalent if and only if they are symmetric around the center of the line. In other words, a line and an equivalence class either do not intersect, intersect only at the center point of the line (which could happen only when m is odd), or intersect at exactly two points. For simplicity, if the elements of A are listed, we avoid the braces when applying any functions to A. For example, we typically write $\phi(3,4)$ instead of $\phi(\{3,4\})$.

2.3 A double-matching

Let $t\ge 4n-2$ be an integer. In what follows, we associate each line in $\mathcal{L}(C_t^n)$ with two of its points in such a way that each point is associated with at most one line. We refer to this association as a *double-matching* for C_t^n. Moreover, our double-matching satisfies that the two points associated with each line are equivalent and different (i.e., they are not the center of the line). We present this matching as a function $f:\mathcal{L}(C_t^n)\to\binom{C_t^n}{2}$, where $f(L)\subset L$ and $f(L)\cap f(L')=\emptyset$ for any two distinct lines $L,L'\in\mathcal{L}(C_t^n)$.

For a given line $L\in\mathcal{L}(C_t^n)$ with vector $\langle l_1,l_2,\ldots,l_n\rangle$, let $A_L=\{l_i: l_i\in\{0,1,\ldots,t-1\}\}$ and $\widehat{A}_L=\{\widehat{l_i}: l_i\in A_L\}$. Then $\widehat{A}_L\subset\{0,1,\ldots,\lceil\frac{t}{2}\rceil-1\}$, and, setting $k=|\widehat{A}_L|$, we have $0\le k\le n-1\le\frac{1}{2}(\frac{t}{2}-1)\le\frac{1}{2}(\lceil\frac{t}{2}\rceil-1)$. Observe that selecting a point $\mathbf{x}_w\in L$ means fixing an index $w\in[0,t-1]$ such that the i^{th} entry of x_w satisfies that

$$x_{w,i}=\begin{cases} l_i & \text{if } l_i\in A_L,\\ w & \text{if } l_i=b,\\ t-1-w & \text{if } l_i=c.\end{cases}$$

The choice of w will be done by means of the extra element assigned to the set $\widehat{A}_L=\{s_1,s_2,\ldots,s_k\}$ with $s_1<s_2<\cdots<s_k$ by the injective function $g_{\lceil\frac{t}{2}\rceil,k}$. More precisely, we define

$$f(L)=L\cap\overline{\mathbf{x}}, \tag{4}$$

where

$$x_i=\begin{cases}\widehat{l_i} & \text{if } l_i\in A_L,\\ 1+s_{\phi(\widehat{A}_L)} & \text{if } l_i\notin A_L.\end{cases}$$

Then the set of different coordinates of $\mathbf{x}$ is equal to $\widehat{A}_L\cup\{1+s_{\phi(\widehat{A}_L)}\}=g_{\frac{t}{2},k}(\widehat{A}_L)$. This means, in particular, that $1+s_{\phi(\widehat{A}_L)}\in\{0,1,\ldots,\lceil\frac{t}{2}\rceil-1\}\setminus\widehat{A}_L$, and that $x_i\in\{0,1,\ldots,\lceil\frac{t}{2}\rceil-1\}$ for all $1\le i\le n$. Hence, if the point $\mathbf{y}$ is one of the two points matched to L, then its entries have the form

$$y_i=\begin{cases} l_i & \text{if } l_i\in A_L,\\ 1+s_{\phi(\widehat{A}_L)} & \text{if } l_i=b,\\ t-1-\left(1+s_{\phi(\widehat{A}_L)}\right) & \text{if } l_i=c,\end{cases} \tag{5}$$

for all $1\le i\le n$, or

$$y_i=\begin{cases} l_i & \text{if } l_i\in A_L,\\ t-1-\left(1+s_{\phi(\widehat{A}_L)}\right) & \text{if } l_i=b,\\ 1+s_{\phi(\widehat{A}_L)} & \text{if } l_i=c,\end{cases} \tag{6}$$

for all $1\le i\le n$. In the proof of Theorem 3, we check that $1+s_{\phi(\widehat{A}_L)}<(t-1)/2$, showing that these two points are indeed different.

Example:
Following up the geometric line $L\in\mathcal{L}(C_{41}^{10})$ defined in Section 1.1, with vector $\langle 21,b,37,6,b,c,6,34,2,10\rangle$, we have that $A_L=\{21,37,6,34,2,10\}$ and $\widehat{A}_L=\{\widehat{21},\widehat{37},\widehat{6},\widehat{34},\widehat{2},\widehat{10}\}=\{2,3,6,10,19\}$ because $\widehat{21}=19$, $\widehat{37}=3$, $\widehat{6}=\widehat{34}=6$, $\widehat{2}=2$ and $\widehat{10}=10$. Applying the function ϕ in Section 2.1 to $S=\widehat{A}_L$, we have $s_0=-1$, $s_1=2$, $s_2=3$, $s_3=6$, $s_4=10$, $s_5=19$, $r=\min\{s_i-2i: 0\le i\le 5\}=\min\{-1,0,-1,0,2,9\}=-1$, and the maximum of the subindices achieving r is $\phi(S)=\phi(\widehat{A}_L)=\max\{0,2\}=2$. This yields $\mathbf{x}=(19,1+s_2,3,6,1+s_2,1+s_2,6,6,2,10)=(19,4,3,6,4,4,6,6,2,10)$, and $f(L)=L\cap\overline{\mathbf{x}}=\{(21,4,37,6,4,36,6,34,2,10),(21,36,37,6,36,4,6,34,2,10)\}=\{\mathbf{x}_4,\mathbf{x}_{36}\}$.

Theorem 3 *The function f is a double-matching of C_t^n for any integers $n\ge 2$ and $t\ge 4n-2$.*

Proof. Using the same notation as the one used above for the definition of the function f, we first argue that f is well-defined. Although, it is clear (by definition) that the points given by (5) and (6) are on the line L, we need to verify that these two points are actually different. This is clear when $1+s_{\phi(\widehat{A}_L)}\neq t-1-(1+s_{\phi(\widehat{A}_L)})$. But these two values could potentially be equal when t is odd and $1+s_{\phi(\widehat{A}_L)}=\frac{t-1}{2}$. We note that this cannot happen due to how $s_{\phi(\widehat{A}_L)}$ is chosen. Indeed, since $\phi(\widehat{A}_L)\le k\le n-1$ and $s_{\phi(\widehat{A}_L)}-2\phi(\widehat{A}_L)\le s_i-2i$ for all $0\le i\le k$, then (using $i=0$) we have

$$s_{\phi(\widehat{A}_L)}\le 2\phi(\widehat{A}_L)-1\le 2k-1\le 2(n-1)-1=2n-3.$$

Moreover, the condition $t \geq 4n-2$ for t odd is equivalent to $t \geq 4n-1$ and thus

$$1 + s_{\phi(\widehat{A}_L)} \leq 2n-2 \leq 2\left(\frac{t+1}{4}\right) - 2 = \frac{t-3}{2} < \frac{t-1}{2}.$$

We now check the two conditions $f(L) \subset L$ and $f(L) \cap f(L') = \emptyset$ for any two distinct lines $L, L' \in \mathcal{L}(C_t^n)$. The first condition is guaranteed by (4). To prove the second condition, suppose that $\mathbf{y} \in f(L) \cap f(L')$ for some point $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in C_t^n$ and for some lines $L, L' \in \mathcal{L}(C_t^n)$ identified with the vectors $\langle l_1, l_2, \ldots, l_n \rangle$ and $\langle l'_1, l'_2, \ldots, l'_n \rangle$, respectively. We prove that $L = L'$. Consider the set $B = \{\widehat{y}_i : 1 \leq i \leq n\}$ and say $|B| = k+1$. By definition of f, we have that $B = g_{\lceil \frac{t}{2} \rceil, k}(\widehat{A}_L) = g_{\lceil \frac{t}{2} \rceil, k}(\widehat{A}_{L'})$. Since $g_{\lceil \frac{t}{2} \rceil, k}$ is injective, then $\widehat{A}_L = \widehat{A}_{L'}$. By definition of g, we have that $B = \widehat{A}_L \cup \{1 + s_{\phi(\widehat{A}_L)}\} = \widehat{A}_{L'} \cup \{1 + s_{\phi(\widehat{A}_{L'})}\}$ and thus $1 + s_{\phi(\widehat{A}_L)} = 1 + s_{\phi(\widehat{A}_{L'})}$. Let $w = 1 + s_{\phi(\widehat{A}_L)} = 1 + s_{\phi(\widehat{A}_{L'})}$ and j be the smallest index i such that $\widehat{y}_i = w$. Note that whenever $y_i \in \{w, t-1-w\}$, we have that $l_i, l'_i \in \{b, c\}$; and when $y_i \notin \{w, t-1-w\}$, we have that $l_i, l'_i \in \{0, 1, 2, \ldots, t-1\}$. More precisely,

$$l_i = l'_i = \begin{cases} y_i & \text{if } \widehat{y}_i \in B \setminus \{w\}, \\ b & \text{if } y_i = y_j, \\ c & \text{if } y_i = t-1-y_j. \end{cases}$$

Therefore, $L = L'$ which concludes the proof. □

Figure 1 shows a visual example of this matching when $n = 3$ and $t = 12$. In this figure, the pairs of red points on the same horizontal line are assigned to that line; the pairs of green points on the same vertical line are assigned to that line; and in general, the pairs of points with the same color c on a geometric line L and in the same direction of a line with color c indicated by the key at the bottom of the figure are assigned to L.

2.4 Balanced colorings from double-matchings

The following lower bound for $\overline{\chi}_b(C_t^n)$ follows from the existence of double-matchings.

Theorem 4 *For integers $n \geq 2$ and $t \geq 4n-2$,*

$$\overline{\chi}_b(C_t^n) \geq \frac{3t^n - (t+2)^n}{2}.$$

Proof. We need to show that there is a balanced $(3t^n - (t+2)^n)/2$ coloring of C_t^n with no rainbow lines. This means that each line must have at least two points of the same color. Note that, by identity (1), we have $(3t^n - (t+2)^n)/2 = |C_t^n| - |\mathcal{L}(C_t^n)|$, and the hypothesis $t \geq 4n-2$ guarantees that $|C_t^n| = t^n \geq 3t^n - (t+2)^n = 2|\mathcal{L}(C_t^n)|$. Because the coloring is balanced, $|\mathcal{L}(C_t^n)|$ colors must be used twice and the remaining $|C_t^n| - 2|\mathcal{L}(C_t^n)|$ colors must be used once. In other words, our coloring must satisfy that each of the $|\mathcal{L}(C_t^n)|$ colors that appear twice blocks a geometric line. To achieve this, we use the same color for the two points assigned to each line in $\mathcal{L}(C_t^n)$ by the function f, using different colors for different lines. The remaining $|C_t^n| - |\mathcal{L}(C_t^n)|$ points use the $|C_t^n| - |\mathcal{L}(C_t^n)|$ colors that appear once. □

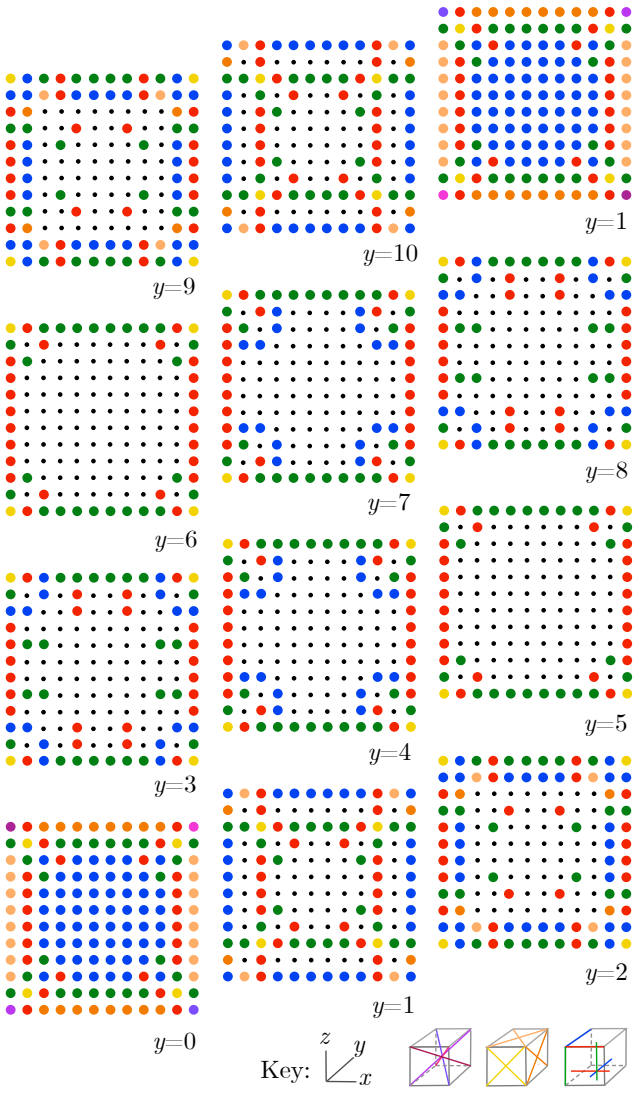


Figure 1: A double-matching for C_{12}^3.

3 Final remarks

Comparing the bounds for t in Proposition 1 and Theorem 4, for $n \geq 2$, we note that $2/(\sqrt[n]{2} - 1) < 4n - 2$. Indeed, $n \geq 2 > 2/(4-e)$ implies that

$$e < \frac{4n-2}{n} = \frac{4}{1 + \frac{1}{2n-1}}.$$

This together with the fact that $(1+\frac{1}{2n-1})^{2n-1} < e$, gives

$$\left(1+\frac{1}{2n-1}\right)^{2n} < 4.$$

Thus $\frac{1}{2n-1} < \sqrt[n]{2}-1$ and so $2/(\sqrt[n]{2}-1) < 4n-2$. Then the best range of t for which we can guarantee the identity in Theorem 2 is $t \geq 4n-2$. For integer values of t such that $2/(\sqrt[n]{2}-1) \leq t < 4n-2$, there is still a possibility that Theorem 2 holds but a different coloring needs to be found. For the remaining values $3 \leq t < 2/(\sqrt[n]{2}-1)$ any coloring of C_t^n with no rainbow geometric lines must use a color at least three times. For every $n \geq 2$, consider the set of integers $J_n := \{\lceil 2/(\sqrt[n]{2}-1)\rceil, ..., 4n-3\}$. For instance, $J_2 = \{5\}$ and $J_3 = \{8, 9\}$. In the plane and in the space Theorem 2 holds for $t \geq 6$ and $t \geq 10$, respectively. We were able to extend the result for values of t in $J_2 = \{5\}$ when $n = 2$, and for values of t in $J_3 = \{8, 9\}$ for $n = 3$.

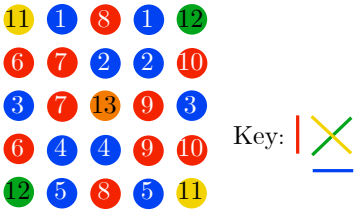


Figure 2: A balanced 13-coloring C_5^2 with no rainbow lines. We use the labels $\{1, 2, \ldots, 13\}$ to indicate the colors, and the corresponding double matching is indicated in red, blue, green, and yellow as shown by the key.

Theorem 5 *In the plane,* $\overline{\chi}_b(C_t^2) = t^2 - 2t - 2$ *for any* $t \geq 5$.

Proof. When $t \geq 4n-2 = 6$, the balanced (t^2-2t-2)-coloring of C_t^2 with no rainbow lines is guaranteed by Theorem 4. The balanced 13-coloring of C_5^2 with no rainbow geometric lines shown in Figure 2 extends the lower bound in Theorem 4 to $t = 5$ when $n = 2$. □

Theorem 6 *In the space,* $\overline{\chi}_b(C_t^3) = t^3 - 3t^2 - 6t - 4$ *for* $t \geq 8$.

Proof. When $t \geq 4n-2 = 10$, the balanced $(t^3 - 3t^2 - 6t - 4)$-coloring of C_t^3 with no rainbow lines is guaranteed by Theorem 4. When $t = 8$, we have that $t^3 - 3t^2 - 6t - 4 = 268$ and the upper bound in Theorem 1 still applies, that is, $\overline{\chi}_b(C_8^3) \leq 268$. However, the double matching in Theorem 3 does not exist. The double-matching in Figure 3 generates a balanced 268-coloring of C_8^3 with no rainbow lines extending the lower bound in Theorem 4 to $t \geq 8$ (we used this matching to find one for $t = 9$). In this matching there are only $t^3 - 2|L(C_8^3)| = t^3 - 2(3t^2 + 6t + 4) = 512 - 488 = 24$ points that are not assigned to lines (uncolored in Figure 3). □

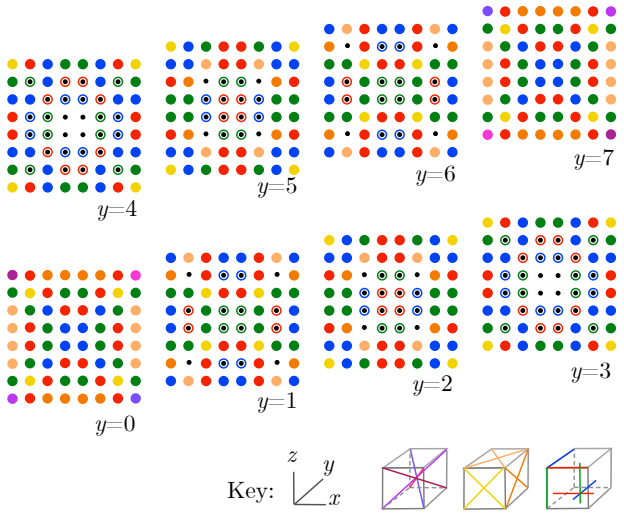


Figure 3: A double-matching for C_8^3. The hollow points are a modification of the general construction for a double-matching of C_t^3 when $t \geq 10$.

We finish by conjecturing the following identity for any dimension.

Conjecture 1 *For integers* $n \geq 2$ *and* $t \geq \frac{1}{\sqrt[n]{2}-1}$, *we have*

$$\overline{\chi}_b(C_t^n) = \frac{3t^n - (t+2)^n}{2}.$$

The question of determining $\overline{\chi}_b(C_t^n)$ for $3 \leq t < 4n-2$ still remains open for higher dimensions and for $4 \leq t \leq 7$ in the space.

References

[1] N. Alon, Y. Caro, Z. Tuza. Sub-Ramsey numbers for arithmetic progressions. *Graphs and Combinatorics*, 5(1):307–314, 1989.

[2] M. Aigner. Lexicographic matching in Boolean algebras. *Journal of Combinatorial Theory Series B*, 14:187—194, 1973.

[3] G. Araujo-Pardo. Daisy structure in Desarguesian projective planes, *J. Australian Mathematical Society*, 74(2):145–154, 2003.

[4] G. Araujo-Pardo, G. Kiss and A. Montejano. On the balanced upper chromatic number of cyclic projective planes and projective spaces. *Discrete Mathematics*, 338(12):2562–2571, 2015.

[5] G. Bacsó and Z. Tuza. Upper chromatic number of finite projective planes. *J. Combinatorial Designs*, 16(3):221–230, 2008.

[6] J. Beck, W. Pegden, and S. Vijay. The Hales-Jewett number is exponential: game-theoretic consequences. In: Analytic Number Theory: Essays in Honour of Klaus Roth. *Cambridge University Press*, 2009.

[7] S. Bhandari and V. Voloshin, Upper chromatic number of n-dimensional cubes, *Alabama Journal of Mathematics*, 43, 2019.

[8] Z. L. Blázsik, A. Blokhuis, Š. Miklavič, Z. L. Nagy, T. Szőnyi. On the balanced upper chromatic number of finite projective planes. *Discrete Mathematics*, 344(3), 2021.

[9] S. Butler, C. Erickson, L. Hogben, K. Hogenson, L. Kramer, R. L. Kramer, J. Chin-Hung Lin, R. R. Martin , Derrick Stolee, N. Warnberg, and M. Young. Rainbow arithmetic progressions. *J. Combinatorics* 7(4):595–626, 2016.

[10] F. R. K. Chung, R. L. Graham. On multicolor Ramsey numbers for bipartite graphs. *J. Combin. Theory Ser. B*, 18 164–169, 1975.

[11] J. A. De Loera, R.N. La Haye, A. Montejano, D. Oliveros, E. Roldán-Pensado. A rainbow Ramsey analogue of Rado's theorem. *Discrete Mathematics*, 339(11):2812–2818, 2016.

[12] P. Erdős, M. Simonovits, V. T. and Sós. Anti-Ramsey theorems. In *Coll. Math. Soc. J. Bolyai*, Vol. 10 of Infinite and Finite Sets, Keszthely (Hungary), pp. 633–642, 1973.

[13] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[14] J. Fox, V. Jungić, R. Radoičić. Sub-Ramsey numbers for arithmetic progressions and Schur triples. *Electronic Notes in Discrete Mathematics*, 28:191–198, 2007.

[15] R. L. Graham, B. L. Rothschild, J. H. Spencer, and J. Solymosi, *Ramsey Theory* (3rd ed.), New York, John Wiley and Sons, 2015.

[16] A. W. Hales, R. I. Jewett. Regularity and positional games. *Trans. Amer. Math. Soc.* 106(2):222–229, 1963.

[17] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10: 26-30, 1935.

[18] M. Huicochea and A. Montejano. The structure of rainbow-free colorings for linear equations on three variables in $\mathbb{Z}_p$, *Integers* 15A(#A8), 2015.

[19] V. Jungić, J. Licht (J. Fox), M. Mahdian, J. Nesetril, and R. Radoičić. Rainbow arithmetic progressions and anti-Ramsey results. *Combinatorics, Probability and Computing*, 12(5–6):599–620, 2003.

[20] V. Jungić and R. Radoičić. Rainbow 3-term Arithmetic Progressions. Integers: The electronic J. Combinatorial Number Theory, 3–A18, 2003.

[21] D. Král'. Mixed hypergraphs and other coloring problems. *Discrete Mathematics*, 307(7-8):923–938, 2007.

[22] A. Montejano. Rainbow considerations around the Hales-Jewett theorem. *ArXiv:2403.13726* [math.CO], 2024.

[23] H. J. Prömel. Complete disorder is impossible: The mathematical work of walter deuber. *Combinatorics, Probability and Computing*, 14(1-2):3–16, 2005.

[24] F. P. Ramsey. A mathematical theory of saving. *The Economic Journal*, 38(152):543–59, 1928.

[25] A. Soifer. Ramsey theory before Ramsey, prehistory and early history: An essay in 13 parts. In *The monograph Ramsey theory yesterday, today and tomorrow*, by A. Soifer (ed.), New York, Springer, 1–26, 2010.

[26] B. L. van der Waerden, Beweis einer Baudeutschen Vermutung. *Nieuw. Arch. Wisk.* 15:212–216, 1927.

[27] V. I. Voloshin. On the upper chromatic number of a hypergraph. *Australasian J. Combinatorics*, 11:25–46, 1995.

Complexity of 2D Snake Cube Puzzles

MIT Hardness Group* Nithid Anchaleenukoon† Alex Dang† Erik D. Demaine† Kaylee Ji†
Pitchayut Saengrungkongka†

Abstract

Given a chain of HW cubes where each cube is marked "turn 90°" or "go straight", when can it fold into a $1 \times H \times W$ rectangular box? We prove several variants of this (still) open problem NP-hard: (1) allowing some cubes to be wildcard (can turn or go straight); (2) allowing a larger box with empty spaces (simplifying a proof from CCCG 2022); (3) growing the box (and the number of cubes) to $2 \times H \times W$ (improving a prior 3D result from height 8 to 2); (4) with triangular prisms rather than cubes, each specified as going straight, turning 60°, or turning 120°; and (5) allowing the cubes to be encoded implicitly to compress exponentially large repetitions.

1 Introduction

Snake Cube [1] is a physical puzzle consisting of wooden unit cubes joined in a chain by an elastic string running through the interior of each cube. For every cube other than the first and last, the string constrains the two neighboring cubes to be at opposite or adjacent faces of this cube, in other words, whether the chain must continue straight or turn at a 90° angle. In the various manufactured puzzles, the objective is to re-arrange a chain of 27 cubes into a $3 \times 3 \times 3$ box.

To generalize this puzzle, we ask: given a chain of DHW cubes, where D, H, W are positive integers, is it possible to rearrange the cubes to form a $D \times H \times W$ rectangular box? We call this problem $D \times H \times W$ Snake Cube. Previous results on its complexity include:

- Abel et al. [1] proved $8 \times H \times W$ Snake Cube is NP-complete by reduction from 3-Partition.
- Demaine et al. [2] proved 2D Snake Cube Packing—deciding whether a chain of cubes can ***pack*** (but not necessarily fill) a $1 \times H \times W$ rectangular box where all cubes are constrained to align with the box—is NP-complete by reduction from Linked Planar 3SAT. This result also holds for a closed chain [2].

Both [1] and [2] pose the (still) open problem of determining the complexity of $1 \times H \times W$ Snake Cube:

Open Problem 1 (2D Snake Cube) *Is* $1 \times H \times W$ Snake Cube *NP-hard?*

1.1 Our Results

In this paper, we prove NP-hardness of several variations of Open Problem 1:

- In Section 4, we prove NP-completeness of 2D Snake Cube with Wildcards where at some cubes there is a free choice between straight or turn. This is motivated by a variant of the snake cube puzzle where a slit cut into a cube allows the chain to continue at a 90° or 180° angle.

  We also give an alternative proof that 2D Snake Cube Packing is NP-complete, simplifying [2].
- In Section 5, we prove that $2 \times H \times W$ Snake Cube is NP-complete. This improves the result of Abel et al. [1] from $D = 8$ to $D = 2$.
- In Section 6, we prove NP-completeness of Hexagonal 2D Snake Cube Packing: deciding whether a chain of hexagonal prisms each specified as going straight, turning 60°, or turning 120° can be packed into a 60°, $H \times W$ parallelogram. Similar to [2], we extend this result to closed chains. One can view this as an improvement to [3] in that angles can be restricted to be in $\{60°, 120°\}$.
- In Section 7, we prove *weak* NP-hardness of 2D Snake Cube, allowing the chain of cubes to be encoded to efficiently represent repeated sequences.

The first three results are reductions from Numerical 3D Matching following a similar framework detailed in Section 3, while the last result is a reduction from 2-Partition. We introduce both base problems in Section 2.

Not all results are proven fully in this paper. All omitted details can be found in the full version of the paper.

*Artificial first author to highlight that the other authors (in alphabetical order) worked as an equal group. Please include all authors (including this one) in your bibliography, and refer to the authors as "MIT Hardness Group" (without "et al.").

†MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {nithidan,alexdang, edemaine,kayleeji,psaeng}@mit.edu

2 Preliminaries

We define our exact problems in mathematical terms.

A ***box*** is the $D \times H \times W$ rectangular cuboid that the cubes of the snake-cube puzzle must fit into. This box can be visualized as a cubic grid where each cube occupies one space of the grid. A ***program*** is a length-k string of ***instructions*** $\mathcal{P} = p_1 \dots p_k$, where each instruction is either the character `T` or `S`. The ***chain*** is the corresponding sequence of adjacent cubes $(c_1, \dots, c_k)$ following the program such that each instruction p_i (where $i \in \{2, \dots, k-1\}$) constrains the angle between the 3 cubes c_{i-1}, c_i, c_{i+1} to be 90° for $p_i =$ `T` (i.e., a turn) and 180° for $p_i =$ `S` (i.e., a straight). A length-k ***segment*** refers to a length-k subchain where all cubes are constrained to form a straight line (e.g., the subchain following the instructions `TSSST` refers to a length-5 segment). If s is a sequence of instructions, let $(s)^k$ denote s repeated k times (e.g., $\texttt{T}(\texttt{ST})^3$ is equivalent to `TSTSTST`). The input to all problems is the box and program. In 2D Snake Cube with Wildcards, each instruction may also be a third character $*$ denoting that the angle can be either 90° or 180°. The instructions in Hexagonal 2D Snake Cube Packing use three different characters introduced in Section 6.

2D Snake Cube with Wildcards, $2 \times H \times W$ Snake Cube, and Hexagonal 2D Snake Cube Packing are in NP, because verification only requires checking all constraints, which takes linear time with respect to the size of the box.

2.1 Reduction Base Problems

Given a multiset $A = \{a_1, a_2, \dots, a_n\}$ of positive integers, ***2-Partition*** is the problem of deciding whether there exists a partition of A into disjoint union $A_1 \sqcup A_2$ such that the sums of elements in A_1 and in A_2 are equal. This problem is known to be weakly NP-hard when the number a_i's are encoded in binary (thus may have exponential value) [4, Subsection A3.2].

For any given target sum t and sequences $(a_i)_{i=1}^n$, $(b_i)_{i=1}^n$, and $(c_i)_{i=1}^n$, each consisting of n positive integers, ***Numerical 3-Dimensional Matching (N3DM)*** is a problem to decide whether there exist permutations σ and π of set $\{1, \dots, n\}$ that satisfies $a_i + b_{\sigma(i)} + c_{\pi(i)} = t$ for all i. This problem is known to be NP-hard even when the numbers are encoded in unary [4, Subsection A3.2]. We refer to a solution to an instance of N3DM as a ***matching***.

Since we can transform an instance of N3DM by setting $a'_i = a_i + 4X$, $b'_i = b_i + 2X$, $c'_i = c_i + X$, and $t' = t + 7X$, for a large integer X (linear in t), the following proposition holds.

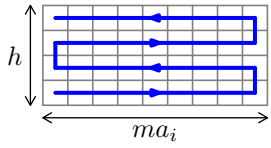
Proposition 2 N3DM *is NP-hard even when we assume that* $a_i \in (0.5t, 0.6t)$, $b_i \in (0.25t, 0.3t)$, *and* $c_i \in (0.125t, 0.15t)$ *for all* $1 \le i \le n$.

3 Overview of Reductions from N3DM

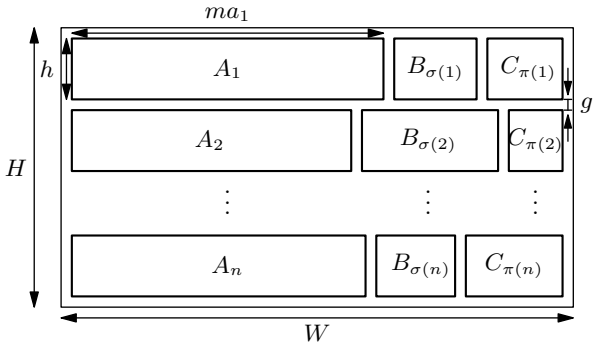
The reductions in Sections 4, 5, and 6 all share a very similar infrastructure, which we informally outline here. In this overview, we let $D = 1$. We explain how to adapt this framework to $D = 2$ in Section 5.

We reduce from the variant of N3DM in Proposition 2. Let $(a_i)_{i=1}^n$, $(b_i)_{i=1}^n$, $(c_i)_{i=1}^n$, and t be an instance of N3DM. We choose the following parameters: the gap width $g = \Theta(n)$, the height of the block $h = \Theta(n^2)$, and the width multiplier $m = \Theta(n^3)$.

The structure of the reduction is as follows. The dimensions of the box are $D \times H \times W = 1 \times (nh + (n+1)g) \times (mt + 4g)$. The numbers $(a_i)_{i=1}^n$, $(b_i)_{i=1}^n$, and $(c_i)_{i=1}^n$ are represented by ***block gadgets*** $(\langle A_i \rangle)_{i=1}^n$, $(\langle B_i \rangle)_{i=1}^n$, and $(\langle C_i \rangle)_{i=1}^n$, which are instructions that can generate ***blocks*** $(A_i)_{i=1}^n$, $(B_i)_{i=1}^n$, and $(C_i)_{i=1}^n$ of dimensions $1 \times h \times ma_i$, $1 \times h \times mb_i$, and $1 \times h \times mc_i$, respectively. Blocks typically consist of h segments as shown in Figure 1a, but details vary in different variants. In the instructions, each block gadget will be separated by a ***wiring gadget***, a sequence of instructions that allows connecting between two adjacent blocks no matter where they are in the grid.



(a) A typical block



(b) The high-level structure of the reduction

Figure 1: The reduction

If a matching exists (i.e., there exist two permutations σ and π of $\{1, 2, \dots, n\}$ such that $a_i + b_{\sigma(i)} + c_{\pi(i)} = t$ for all i), then (ignoring the wiring gadget) one can arrange the blocks into a perfect $1 \times nh \times mt$ rectangle by aligning each triple of blocks A_i, $B_{\sigma(i)}$, and $C_{\pi(i)}$ together in the same row. Since our box is slightly larger than $1 \times nh \times mt$, we can place the blocks such that there is a gap g between neighboring blocks and between each block and the boundary of the rectangular box. The gap g is chosen so there is sufficient space

for a subchain following the wiring gadget to connect all the blocks. Wires detour around blocks and do not cross; the explicit algorithm will be given in Lemma 4. Finally, depending on the variant, there may be additional instructions at the end of the program to fill in the remaining space in the box. Figure 1b depicts the overall reduction structure.

In the other direction, we also need to show that the existence of a chain following the program forces the existence of matching, even if the block gadgets $\langle A_i \rangle$, $\langle B_i \rangle$, and $\langle C_i \rangle$ do not fold into perfectly aligned and evenly spaced blocks (e.g., if part of a subchain following $\langle B_i \rangle$ may go into gaps between subchains following $\langle A_i \rangle$). In the following subsection, we prove Lemma 3 that shows the existence of a chain following the program necessitates the existence of a matching, even if blocks do not fold ideally.

3.1 Segment Packing Lemma

We view each block as h segments; for instance, the block gadget $\langle A_i \rangle$ specifies h consecutive ma_i-segments. Thus, we have $3nh$ segments, h of each length $ma_1, \ldots, ma_n, mb_1, \ldots, mb_n, mc_1, \ldots, mc_n$ to pack into the box. This motivates the following "Segment Packing Lemma".

Lemma 3 (Segment Packing Lemma) *Let $(a_i)_{i=1}^n$, $(b_i)_{i=1}^n$, $(c_i)_{i=1}^n$, t be an instance of* N3DM *satisfying the conditions in Proposition 2. Let m and h be positive integers, and consider a $1 \times H \times W$ box where $W > mt$ and $nh < H < m$. Suppose there are $3nh$ segments of $3n$ types $A_1, \ldots, A_n$, $B_1, \ldots, B_n$, and $C_1, \ldots, C_n$. If all of the following are true, then there exists an N3DM matching:*

- *$W < m(t+1)$ and $H < nh + \frac{h}{40}$;*
- *for all $1 \le i \le n$, all segments of type A_i, B_i, and C_i have lengths ma_i, mb_i, and mc_i, respectively;*
- *there are exactly h segments of each type; and*
- *no two segments of the same type are more than h rows vertically apart (note that since $m > H$, all $3nh$ segments must lie horizontally in the box.).*

Proof. (Sketch) We call ma_i-segments A-segments, and analogously for B-segments and C-segments. From constraints in Proposition 2, each row of the box must be of one of the following four categories: (1) a *good row*, which contains exactly one A-segment, one B-segment, and one C-segment; (2) an *A-bad row*, which contains no A-segment; (3) a *B-bad row*, which contains one A-segment but contains no B-segment; and (4) a *C-bad row*, which contains one A-segment, one B-segment, but no C-segment. Let n_{good}, n_A, n_B, and n_C denote the number of good rows, A-bad rows, B-bad rows, and C-bad rows, respectively. Due to the constraints of Proposition 2 and $W < m(t+1)$, we count the number of A-, B-, and C-segments to derive the following inequalities:

$$\begin{aligned} n_A &= H - nh < \tfrac{h}{40} \\ n_B &\le 2n_A + \tfrac{h}{40} < \tfrac{3h}{40} \\ n_C &\le 6n_A + 2n_B + \tfrac{h}{40} < \tfrac{13h}{40}. \end{aligned}$$

Therefore, $n_A + n_B + n_C < h$.

Finally, we color each row by its residue modulo h. Thus, there are either n or $n+1$ of each color. Moreover, there exists color c that colors only good rows. Since segment of the same type are less than h rows apart, there is exactly one segment of each type colored c and exactly n rows of color c. For each row of color c, let ma_i, mb_j, and mc_k be the segment lengths. Then,

$$ma_i + mb_j + mc_k \le W < m(t+1) \implies a_i + b_j + c_k \le t.$$

Summing the inequality for each row of color c gives $nt \le nt$, so all inequalities must be equalities. Therefore, $a_i + b_j + c_k = t$ for each row of color c, forming a solution to the instance of N3DM. □

3.2 Connecting Wires

This subsection concerns the wiring part. It guarantees that, if the gap is large enough, there exists a way to place wiring gadgets without crossing, regardless of the arrangement of blocks forced by a solution to the instance of N3DM. This lemma was adapted from [5, Lemma 5].

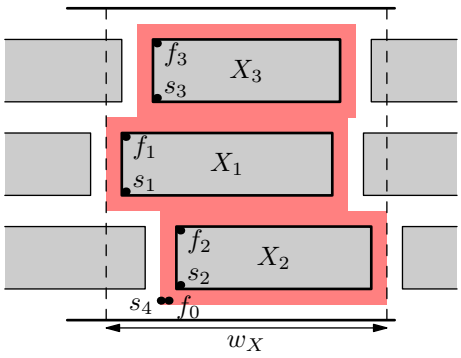


Figure 2: Example of the setup for wire packing when $n = 3$. Red area represents available space.

The setup for this lemma is depicted in Figure 2 and goes as follows: given a bounding box of size $H' \times w_X$ and locations of ***rectangles*** $X_1, X_2, \ldots, X_n$ with widths $x_1, x_2, \ldots, x_n$, respectively, and the same height h'. Each row contains at most one rectangle, but the rectangles are in arbitrary order from top to bottom. Note that the "rectangles" are not the same as blocks; a

rectangle consists of squares, and a square is filled with 2×2 cubes which will be discussed further in Section 4 when applying this lemma to prove the existence of a chain. Define a wire connecting squares a and b to be a sequence of adjacent squares with the first and the last squares are adjacent to a and b, respectively.

Lemma 4 (Wire Lemma) *Assume the above setup with $\min_i x_i > w_X/2$, and all rectangles are at least $g' \geq 100n$ squares apart. Define the **available space** to be a set of squares in the extension of all rectangles on each edge by $g'/2$. For each $i = 0, 1, \ldots, n$, let ℓ_i be an even integer in $[8nw_X, 12nw_X]$. Let s_i and f_i be the bottom-left and top-left corners of rectangle X_i, and f_0, s_{n+1} be two chosen squares at the bottom-left of the available space. Then, one can draw $n+1$ disjoint wires $W_0, \ldots, W_n$ in the available space, where W_i has length exactly ℓ_i and W_i connects f_i to s_{i+1} for all $i \in \{0, 1, \ldots, n\}$. Furthermore, no two cells from different wires W_i and W_j are adjacent.*

Proof. (Sketch) We will briefly explain an algorithm to place the wires $W_0, \ldots, W_n$ inductively. First, mark squares $m_0 = f_0, m_1, \ldots, m_n, m_{n+1} = s_{n+1}$ in the same row in this order; all of these should be near the bottom-left of overall available space. We will construct wires $(U_i)_{i=1}^n$ and $(V_i)_{i=1}^n$ such that U_i connecting m_{i-1} to s_i, and V_i connecting m_i to f_i. Then, W_i is a concatenation of wire U_{i+1}, square m_i, and wire V_i for all $i \in \{1, \ldots, n-1\}$. Moreover, $W_0 = U_0$ and $W_n = V_n$. We also reserve space of width $40n$ squares above and below each rectangle and $10n$ squares on the left of each rectangle. The two main stages of placing wires are

(a) Place U_i and V_i without crossing U_1, V_1, …, U_{i-1}, V_{i-1}. This process is done inductively.

(b) Adjust the length of the wire W_i to be exactly ℓ_i by placing the remaining length U_i and V_i inside reserved space of rectangle X_i, which has size at least $40n \times x_i$; the space can fit a wire of length $> 20nw_X$, large enough to contain the extra length.

To accomplish (a), place U_i and V_i by following these steps simultaneously for each i.

(i) Create a sequence of squares from m_i to the top of the available space, following along the left gaps.

(ii) Draw the wire down to the same row as s_i between the wires we have placed in (i) and the left edges of all rectangles, and then draw the wire horizontally to s_i.

(iii) The current wire may cross U_j or V_j for some $j < i$ when they are horizontally connected to s_j or f_j. In this case, replace the current wires by making then go around other edges of rectangle X_j.

To justify the size of available space, each of U_i and V_i may contribute to at most 2 layers of wires on each edge of the block with a space of one square between each layer of wires. Combine this with the reserved space; we need available space with width $40n + 2 \cdot 2 \cdot (2n) < 50n$ on each edge of the rectangles.

The dominant contribution to the length of the wire occurs when the wires have to go around other rectangles since $w_X \gg nh' + (n+1)g'$. However, there are at most n blocks that a wire has to go around. Including all other distances, the sufficient length of a wire is $8nw_X$. □

4 Snake Cube Puzzles in $1 \times H \times W$ box

In this section, we consider the 2-dimensional variants of Snake Cube. We first consider 2D Snake Cube with Wildcards, where we allow the wildcard `*` that could be used as either `S` or `T`. We will prove the following:

Theorem 5 2D Snake Cube with Wildcards *is NP-hard.*

Subsection 4.1 will sketch the proof of Theorem 5. Then, in Subsection 4.2, we will explain how to modify this proof to give an alternative proof of the following, which was first proved in [2].

Theorem 6 2D Snake Cube Packing *is NP-hard.*

4.1 Proof with Wildcard Option

Given an instance of N3DM with target sum t, $(a_i)_{i=1}^n$, $(b_i)_{i=1}^n$, $(c_i)_{i=1}^n$, where $a_i \in (0.5t, 0.6t)$, $b_i \in (0.25t, 0.3t)$, and $c_i \in (0.125t, 0.15t)$ for all i (Proposition 2), we define these parameters to construct a string input to 2D Snake Cube with Wildcards.

$$\begin{aligned} g = &\quad \text{gap width} &&= 200n \\ m = &\quad \text{multiplier of widths} &&= 30000n^3 \\ h = &\quad \text{height of blocks} &&= 20000n^2 \\ H = &\quad \text{height of the grid} &&= nh + (n+1)g \\ W = &\quad \text{width of the grid} &&= mt + 4g \end{aligned}$$

Then, construct block gadgets A_i, B_i, and C_i for all $1 \leq i \leq n$. The sequence for A_i is given below, and the sequences for B_i, C_i are analogous. These blocks will fold into rectangles of size $h \times ma_i$, $h \times mb_i$, or $h \times mc_i$.

$$\langle A_i \rangle = (\mathtt{S})^{ma_i - 1}(\mathtt{TT}(\mathtt{S})^{ma_i - 2})^{h-1}$$

The program is given by

$$\begin{aligned} &\langle A_1 \rangle (*)^{16nW} \langle A_2 \rangle (*)^{16nW} \ldots (*)^{16nW} \langle A_n \rangle (*)^{16nW} \\ &\langle B_1 \rangle (*)^{8nW} \langle B_2 \rangle (*)^{8nW} \ldots (*)^{8nW} \langle B_n \rangle (*)^{8nW} \\ &\langle C_1 \rangle (*)^{4nW} \langle C_2 \rangle (*)^{4nW} \ldots (*)^{4nW} \langle C_n \rangle (*)^{4nW} (*)^{\ell}, \end{aligned}$$

where the number ℓ of $*$'s at the end is to make the length of the whole program exactly WH.

Chain ⇒ Matching. Each block gadget $\langle A_i \rangle$, $\langle B_i \rangle$, $\langle C_i \rangle$ contains h segments that must be horizontal due to its length. Thus, we have $3nh$ segments of $3n$ types that fit the condition of Lemma 3. The lemma forces the existence of matching.

Matching ⇒ Chain. Suppose there is a matching, so we can place all blocks as shown in Figure 1b.

Apply the wire lemma to connect all these blocks where each square in the lemma corresponds to 2×2 cubes in this construction. Hence, each block is at least $g' = g/2 = 100n$ squares apart, and the width of the bounding box is $w_X = W/2$ squares. For wires among A_i blocks, Lemma 4 implies that there exists a sequence of $8nw_X = 4nW$ squares of size 2×2 that connect all of A_i blocks. These squares can be filled with $16nW$ $*$ cubes as demonstrated in Figure 3. For wires among B_i blocks, the block size is roughly half of A_i; thus, we can reduce the parameter w_X in the lemma by half. Similarly, the parameter is reduced to a quarter for C_i blocks.

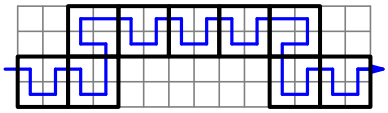


Figure 3: Example of filling a wire with $*$ tiles.

The wire between A_n and B_1 is long enough to connect the following cubes in order: (1) the last cube of block A_n, (2) the cube at the square marked as f_n in the application of Lemma 4 to connect wires between A_i blocks, (3) the cube at the square marked as s_0 in the application of Lemma 4 to connect wires between B_i, and (4) the first cube of block B_1. This is always possible since gaps are all connected, and the length needed never exceeds $8nW + 4W + 4nW \leq 16nW$. The wire between B_n and C_1 can be placed similarly.

Lastly, notice that the construction we described so far is aligned with the 2×2 polygrid, and the remaining squares are connected because the blocks and wires are topologically equivalent to a path with no closed loop. From [6], there exists a Hamiltonian cycle in any connected shape aligned with 2×2 polygrid. Thus, the remaining wildcards can fill all the remaining space.

4.2 Proof Outline for Packing

To prove Theorem 6, the setup and the main proof are almost the same. The only difference is that we cannot use the wildcard $*$. To fix this, we make two modifications. First, we remove $(*)^\ell$ at the end because we do not need to fill the box. Second, we replace all $*$'s between block gadgets with an equally long string of T's. By making squares in Lemma 4 correspond to 2×2 cubes, wiring gadgets can connect different block gadgets using only T's. This is possible because not all cubes in the wires need to be used, and cubes outside the wires may be used. We can make the wire length exactly as specified by varying how the chain fills the wires at turns. More details are available in the full version of the paper.

5 Snake Cube Puzzles in $2 \times H \times W$ box

In this section, we outline our reduction from N3DM to $2 \times H \times W$ Snake Cube. A detailed proof can be found in the full version of the paper.

Theorem 7 *$2 \times H \times W$ Snake Cube is NP-hard.*

We follow the block and wire reduction infrastructure introduced in Section 3. Consider an instance of N3DM $(a_i)_{i=1}^n$, $(b_i)_{i=1}^n$, $(c_i)_{i=1}^n$, and t satisfying the conditions in Proposition 2. Define the parameters as follows.

$$g = 1200n, \quad h = 60000n^2, \quad m = 60000n^3,$$
$$H = n(h + 6g + 4) + 2, \quad W = 4g + mt + 6.$$

In this variant, the blocks generated by the block gadgets have depth 2 (e.g., block A_i is of size $2 \times h \times ma_i$). Like previously, the program consists of block gadgets separated by wire gadgets. However, unlike previous $1 \times H \times W$ variants, a second layer results in fewer constraints on the shape of subchains following the block gadget, so Lemma 3 no longer applies. To bypass this issue, we introduce additional instructions at the beginning of the program that specifies a ***shelf*** — the structure shown in Figure 4 that constrains the folding of the $\langle A_i \rangle$, $\langle B_i \rangle$, and $\langle C_i \rangle$ subchains. The shelf is designed so that it can only be made into a subchain the intended way. The program is given by

$$\langle \text{shelf} \rangle \langle A_1 \rangle (\texttt{T})^{96nW} \langle A_2 \rangle (\texttt{T})^{96nW} \dots (\texttt{T})^{96nW} \langle A_n \rangle (\texttt{T})^{96nW}$$
$$\langle B_1 \rangle (\texttt{T})^{48nW} \langle B_2 \rangle (\texttt{T})^{48nW} \dots (\texttt{T})^{48nW} \langle B_n \rangle (\texttt{T})^{48nW}$$
$$\langle C_1 \rangle (\texttt{T})^{24nW} \langle C_2 \rangle (\texttt{T})^{24nW} \dots (\texttt{T})^{24nW} \langle C_n \rangle (\texttt{T})^{\ell},$$

where $(\texttt{T})^\ell$ pads the string to length $2HW$ and $\langle A_i \rangle = (\texttt{S})^{ma_i - 1} (\texttt{TT}(\texttt{S})^{ma_i - 2})^{2h-1} \texttt{S}$; $\langle B_i \rangle$ and $\langle C_i \rangle$ are defined analogously.

Chain ⇒ Matching. Since the gap to the right of each shelf is small ($g \ll m$), all cubes within a block must fit entirely within one row of the shelf. By a similar counting argument as in Lemma 3, there exists a row containing exactly one A_i, B_j, and C_k block in each shelf, and the corresponding a_i, b_j, and c_k must sum to t.

Matching ⇒ Chain. Place all blocks as in Figure 1b where segments in the blocks fill the top and bottom layers alternately. The remaining grid can be partitioned into $2 \times 4 \times 4$ blocks of space. We cite a result from [1]:

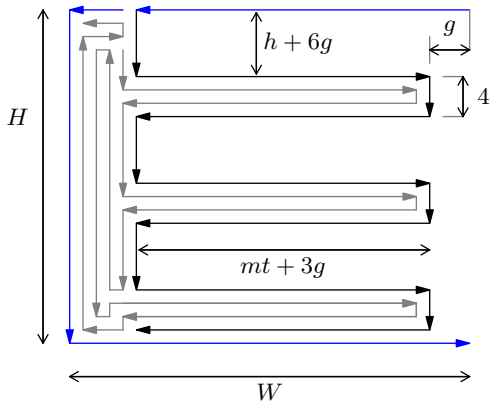


Figure 4: One layer of the "shelf" with 3 rows. The chain moves to the other layer at discontinuities.

given a sequence $(\mathtt{T})^8$ of cubes entering a $2 \times 2 \times 2$ block of space, the cube chain can exit from any face. Thus, traversing between $2 \times 2 \times 2$ blocks of space with $(\mathtt{T})^8$'s has the same movement freedom as traversing between cells in a 2D grid with wildcards. Thus, by grouping $2 \times 2 \times 2$ cubic blocks together, the remaining proof is equivalent to that in Section 4, except the wires are $\frac{3}{2}$ times long to allow for detours around the shelf.

6 Snake Cube Puzzles with Hexagonal Prisms

In this section, we consider a version of a 2D Snake cube with a chain of hexagonal prisms. When the prisms are represented by points, the movement patterns form a triangular grid. Thus, the problem becomes a triangular grid variant of the flattening fixed-angle chains problem in [2].

An infinite ***triangular grid*** is a two-dimensional lattice generated by vectors $v_1 = \binom{1}{0}$ and $v_2 = \binom{\cos 60^\circ}{\sin 60^\circ}$; each point represents a hexagonal prism. Two points in a triangular grid are ***adjacent*** if they are distance 1 apart. A **60° *parallelogram box*** of dimension $H \times W$ is the set of HW points obtained by translating the set $\{iv_1 + jv_2 : i \in \{1, \ldots, W\}, j \in \{1, \ldots, H\}\}$ by some lattice vector.

For this section, a ***program*** is a string that consists of only characters $\mathtt{S}$, $\mathtt{T}_{60}$, and $\mathtt{T}_{120}$, where $\mathtt{S}$ denotes straights, $\mathtt{T}_{60}$ denotes 60° turns (forming 120° angle), and $\mathtt{T}_{120}$ denotes 120° turns (forming 60° angle). We say that a chain $C = (p_1, p_2, \ldots, p_{|s|})$ (length $|s|$) satisfies s if and only if for every $i \in \{2, 3, \ldots, |s| - 1\}$, the angle between p_{i-1}, p_i, p_{i+1} is 180° if $s_i = \mathtt{S}$, 60° if $s_i = \mathtt{T}_{120}$, and 120° if $s_i = \mathtt{T}_{60}$. C is ***closed*** if and only if $p_1 = p_{|s|}$.

Theorem 8 *Both of the following problems are NP-complete.*

- Bounded Triangular Path Packing*: given a 60° parallelogram box B, a program $\mathcal{P}$, and two adjacent vertices u and v on a boundary of B, decide whether there is a chain connecting u and v satisfying $\mathcal{P}$.*
- Triangular Closed Chain*: given a program $\mathcal{P}$, decide whether there is a closed chain satisfying $\mathcal{P}$.*

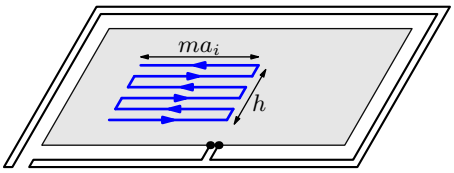


Figure 5: The frame gadget and an example block gadget inside.

To prove the first problem NP-Hard, we use the same reduction, except that block gadgets are 60° parallelograms shown in Figure 5. Then, we can reduce the first problem to the second problem, creating a ***frame*** gadget to force the chain by modulo a large prime condition similar to [2] shown in Figure 5.

7 Weak-NP-hardness of 2D Snake Cube Puzzle

In this section, we consider 2D Snake Cube, where the chain must fill a $1 \times H \times W$ rectangle. However, we allow the instructions to be encoded using the shorthand notation, which keeps the inputs polynomial with respect to the input integers. Since this modification means the problem may no longer be in NP, this reduction only proves NP-hardness. For any set S, let $\sum S$ be the sum of its elements.

Let A be the multiset of positive integers, a 2-Partition instance. We select $H = 2|A| + 4$ and $W = 4\sum A + 1$. The program comprises the ***caps*** at either end and $|A|$ ***layers*** in between, encoding each a_i in A sequentially. The ***swivel points*** join each gadget and allow the layers to flip horizontally. The orientation of each layer left or right corresponds to assigning each a_i to either partition (Figure 6).

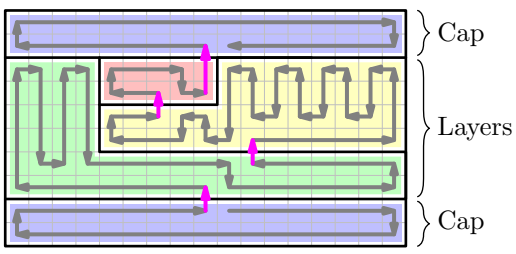


Figure 6: Chain for $A = \{1, 2, 1\}$, emphasizing the different gadgets, highlighting the swivel points (in bolded red), and demonstrating the 3 variants of layers.

7.1 Gadgets

The starting cap is the subsequence (the ending cap being the reverse):

$$(\mathtt{S})^{\frac{W-1}{2}-1}\mathtt{TT}(\mathtt{S})^{W-2}\mathtt{TT}(\mathtt{S})^{\frac{W+1}{2}-1}\mathtt{T}\ldots.$$

Since $W > H$, the W-segments in the caps can only fit horizontally. They must be at the top and bottom since any other position would create an unfillable empty space. This forces the position of the swivel points joining the caps and the layers to be horizontally centered.

For each i, let $A_i = \{a_j : j \in \{1, \ldots, i\}\}$, $w_i = 4\sum(A \setminus A_{i-1})+1$, $x_i = (w_i-1)/2$, and $h_i = 2|A \setminus A_{i-1}|$. There are 3 variants of the corresponding layer gadget.

If $4a_i \le x_i$ and $h_i > 2$, the layer is the subsequence (sections named for ease of discussion, see Figure 7):

$$\begin{aligned} \ldots\mathtt{T}(\mathtt{S})^{x_i-1}\mathtt{T} &\quad \text{“arm”} \\ (\mathtt{S})^{h_i-2}(\mathtt{TT}(\mathtt{S})^{h_i-3})^{4a_i-1}\mathtt{T} &\quad \text{“padding”} \\ (\mathtt{S})^{x_i-4a_i+1}(\mathtt{T})^{2(2a_i-1)} &\quad \text{“shift”} \\ (\mathtt{S})^{x_i-2a_i}\mathtt{TT}(\mathtt{S})^{x_i-2a_i+1}\mathtt{T}\ldots &\quad \text{“return.”} \end{aligned}$$

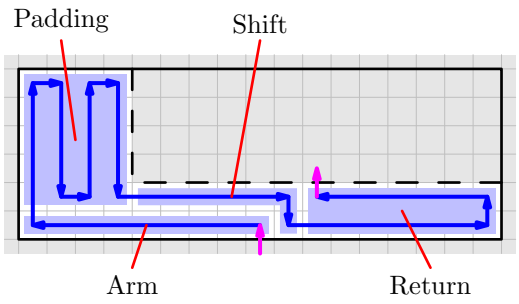


Figure 7: Sample layer gadget with $a_i = 1$, $w_i = 17$, $h_i = 6$ with labeled sections.

If $4a_i > x_i$ and $h > 2$, informally the padding *spills over* into the shift, resulting in these differences:

$$\begin{aligned} (\mathtt{S})^{h-2}(\mathtt{TT}(\mathtt{S})^{h-3})^{x_i}(\mathtt{TT}(\mathtt{S})^{h-2})^{4a_i-x_i-1} &\quad \text{“padding”} \\ (\mathtt{T})^{2(2a_i-1-(4a_i-x_i-1))} &\quad \text{“shift.”} \end{aligned}$$

If $h = 2$, informally the padding can be visualized as degenerating and subsuming the shift and return, resulting in these changes from the first variant:

$$\mathtt{T}(\mathtt{S})^{x_i}(\mathtt{T})^{2(2a_i-1)+1}\mathtt{S} \qquad \text{“padding.”}$$

Each layer gadget has a $w_i \times h_i$ space available to it and leaves behind a $w_{i+1} \times h_{i+1}$ space while displacing the swivel point horizontally by $2a_i$ left or right. To show this, we use induction starting from the first layer. Note that the arm and padding sections are all forced by space constraints. The shift section is forced since turning the chain outward in the subsequence of repeated turns (T) would leave behind a 1×1 space. This space can only be filled by the endpoints, which is impossible because their positions are forced by the cap gadgets. Then, the return section is also forced.

7.2 Reduction

If there exists a solution to 2-Partition, then construct all the gadgets and flip the layer gadgets so that arms for all numbers in A_1 point to the left, and those for numbers in A_2 point to the right. The horizontal displacements of the swivel points must sum to 0, so the last layer can connect to the upper cap.

If there exists a solution for 2D Snake Cube, then we have demonstrated the gadgets are forced to be constructed in the correct orientation. Since the last layer gadget connects to the upper cap gadget, the horizontal displacements of the swivel points must sum to 0. Reversing the above process produces a solution to the 2-Partition instance.

Acknowledgement

This work was conducted during open problem-solving sessions in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.5440) in Fall 2023. We thank the other participants in the class—particularly Papon Lapate, Benson Lin Zhan Li, and Kevin Zhao—for related discussions and for providing an inspiring atmosphere.

References

[1] Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, and T. B. Schardl, “Finding a Hamiltonian path in a cube with specified turns is hard,” *Journal of Information Processing*, vol. 21, no. 3, pp. 368–377, 2013.

[2] E. D. Demaine, H. Ito, J. Lynch, and R. Uehara, “Computational complexity of flattening fixed-angle orthogonal chains.” arXiv:2212.12450, 2022. https://arXiv.org/abs/2212.12450. Preliminary version in CCCG 2022.

[3] E. D. Demaine and S. Eisenstat, “Flattening fixed-angle chains is strongly NP-hard,” in *Proceedings of the 12th Algorithms and Data Structures Symposium (WADS 2011)*, pp. 314–325, August 15–17 2011.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[5] Z. Abel and E. Demaine, “Edge-unfolding orthogonal polyhedra is strongly NP-complete,” in *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG 2011)*, 2011.

[6] K. C. Cheung, E. D. Demaine, J. R. Bachrach, and S. Griffith, "Programmable assembly with universally foldable strings (moteins)," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 718–729, 2011.

On Erdős-Szekeres Maker-Breaker games*

Arun Kumar Das† Tomáš Valla‡

Abstract

The *Erdős-Szekeres Maker-Breaker game* is a two-player competitive game where both players alternately place points in the plane such that no three points are colinear. The first player (*Maker*) starts the game by placing her point and wants to obtain an empty convex polygon of a given size k such that the vertices of the polygon are chosen from these points and the second player (*Breaker*) wants to prevent it. We show that Maker wins the game for $k \leq 8$. We also present a winning strategy for Maker for any k in general when Maker is allowed to place $(1 + \varepsilon)$ times more points (each round on average) in comparison to Breaker, for any $\varepsilon > 0$. Further, we address the models of the game for equilateral empty convex polygons in the plane and empty convex polygons in square grids.

1 Introduction

One of the most well-known problems in discrete geometry is the *Erdős-Szekeres Problem* [6]. In this problem, a positive integer k is given as input and we compute the minimum number of points required in the plane such that at least k out of them are in convex position. The points must be placed in *general position*, i.e. no three points are colinear. Erdős and Szekeres showed that the answer is finite by proving that the number is bounded from above by a function exponential in k. Further Erdős [5] posed the problem of finding the minimum number of points in the plane in general position (no three points are colinear) such that k points out of them can be chosen as the vertices of a convex polygon that does not contain any other point from the point set inside it. The empty convex polygon with k vertices is referred to as a *k-hole*. By $H(k)$ we denote the minimum number of points such that any configuration of $H(k)$ points in general position contains a k-hole. Horton [10] demonstrated that there are arbitrary large point sets that do not contain a 7-hole, in contrast to the finiteness of the previous question. For k-holes with $k < 7$, positive results are present in the literature showing $H(5) = 10$ and $H(6) = 30$ [7, 9].

Competitive games between two players to achieve a geometric structure we studied previously [8]. The natural competitive game arising from the Erdős-Szekeres problem is the endeavor of two players to achieve a k-hole for a given positive integer k by alternately placing points in general position in the plane. Depending on the goal of the game, three variants of the two-player game spawn from the Erdős-Szekeres problem.

1. Both players want to obtain a k-hole (*Maker-Maker*). Whoever obtains the k-hole first is the winner.
2. Both players want to avoid a k-hole (*Avoider-Avoider*). Whoever obtains the k-hole first is the loser.
3. The first player wants to obtain a k-hole and the second wants to prevent it (*Maker-Breaker*).

Valla [12] posed the Maker-Maker variant of the game as an open problem in his thesis. Kolipaka and Govindarajan [11] studied the Avoider-Avoider variant. They proved that the game with $k = 5$ ends after round 9 and the second player wins.

Later Aichholzer et al. [1] simplified the original proof by Kolipaka and Govindarajan, and also introduced a different variant of the game with colors, referred to as *bi-chromatic* variant. Here the players alternately place points in general position in the plane, the first player placing red points and the second player placing blue points. Aichholzer et al. [1] showed the winning strategy for the second player for $k = 3$ for the Avoider-Avoider version, where the players try to avoid a monochromatic k-hole. Then they introduced the *Maker-Maker* variant of the bi-chromatic game where both the players try to obtain a monochromatic k-hole. Besides considering the k-holes, they considered the non-convex *general holes* as well. The general hole of size k is an empty simple polygon with k vertices. Aichholzer et al. [1] also showed that the first player can win for $k = 5$ in 9 turns in the Maker-Maker variant. Further, they studied the *Maker-Breaker* variant of the bi-chromatic game where the first player (Maker) wants to obtain a k-hole with only *red* vertices, and the second player (Breaker) just wants to prevent it by placing blue vertices. For this variant, Aichholzer et al. proved that the Maker wins by placing 8 points of her color for a 5-hole and has a

*This work was supported by the Czech Science Foundation Grant no. 24-12046S.

†Faculty of Information Technology, Czech Technical University arun.kumar.das@fit.cvut.cz

‡Faculty of Information Technology, Czech Technical University tomas.valla@fit.cvut.cz

general winning strategy for general holes of any given k.

In this paper, we study the *Maker-Breaker* variant of the Erdős-Szekeres type game (*ESMB*) in the plane. For notational brevity, we name two players Alice and Bob. Both of them place one point in each round alternatively on $\mathbb{R}^2$ maintaining the general position for all the points throughout the game. Alice tries to obtain a k-hole of a given size k. Alice wins the game if she can obtain the k-hole and the game ends after a finite number of moves. Otherwise, we conclude that Bob wins the game if he can always restrict Alice from forming a k-hole. In our model, unlike the bi-chromatic variant, Alice can use any point of her choice to form the empty convex polygon of the desired size. To the best of our knowledge, this variant was not studied previously and only was pointed out as interesting and challenging for $k \geq 7$ by Aichholzer et al[1].

We note that Alice can win the game for a given k in r rounds if there is an empty $(k-1)$-gon at the end of the $(r-1)^{\text{th}}$ round. Alice can extend this existing hole by placing one point very closely without violating the convexity of the newly formed hole. Thus, it can be concluded from the existing literature proving $H(6) = 30$ [9] that Alice wins the game up to $k \leq 7$. But we show that the minimum number of points required for Alice to win is much less than $H(k)$. Further, we show Alice has a winning strategy for $k = 8$ even if $H(7)$ could be arbitrarily large [10]. Then we prove that Alice can win the game for any k if the ratio of the number of points placed by Alice and the points placed by Bob in each round is $(1+\varepsilon)$, for any small $\varepsilon > 0$.

Then we address the question of obtaining equilateral holes. This question has not been considered before and only makes sense in terms of the game as there could be arbitrarily large point sets such that all the distances of point pairs are different. We prove that Alice can obtain an equilateral k-hole for $k = 4$.

Finally, we address the variant of the game on grids. The Erdős-Szekeres problem has been extensively studied concerning the position of points approximating the integer lattice [4, 14]. We consider the Maker-Breaker game on a square grid of size $n \times n$. Both players must place their points at the gridpoints. Considering this constraint, in this version the players are allowed to violate the general position requirement. We characterize the winning strategy of both players depending on the size of n and k.

1.1 Results

We formally state the results as follows.

Theorem 1 *Alice can win the ESMB game by obtaining a 7-hole from 15 points in the 8^{th} round.*

Theorem 2 *Alice can win the ESMB game by obtaining an 8-hole from 25 points in the 12^{th} round.*

Theorem 3 *Alice can win the ESMB game by obtaining a k-hole for any given positive integer k if the ratio of the points placed by Alice to the points placed by Bob is $(1+\varepsilon)$ for any $\varepsilon > 0$.*

Theorem 4 *Alice wins the ESMB game for an equilateral 4-hole by obtaining it from 7 points in the 4^{th} round.*

Theorem 5 *Let us consider the ESMB game on an $n \times n$ square grid where both the players have to place their points on one of the vertices of the grid and they are allowed to place three or more colinear points. Alice can win the game by obtaining a k-hole for any positive integer $k > 2$, if and only if $n \geq \lceil \frac{k}{2} \rceil$.*

1.2 Organization

The paper presents the winning strategies for Alice as stated in Theorem 1, 2 and 3 in Section 2. Section 3 contains the study of equilateral holes for the ESMB game in the plane and Section 4 contains the results for the game on the square grid. Finally, the paper is concluded in Section 5 presenting a list of open problems.

2 Winning strategies for Maker

We start with the formation of a 5-hole in the ESMB game, which is a winning strategy for Alice when $k = 5$.

Lemma 6 *Alice wins the game by obtaining a 5-hole from 7 points in the 4^{th} round.*

Proof. Alice trivially obtains a 3-hole with 3 points in the 2^{nd} round. Then, Bob must place a point that is not in a convex position with the other three to prevent Alice from winning in the immediate next round. Now Alice places A_3 in such a way that she creates one or more 4-holes. We can follow that one 4-hole remains in the plane irrespective of the placement of B_3. Alice extends it to a 5-hole by placing A_4 accordingly. □

Now we show the winning strategy of Alice in the ESMB game for $k = 6, 7$, and 8 by assuming that the game starts with a $(k-1)$-hole and Bob places his point as B_0 followed by Alice's point A_1 and so on.

Lemma 7 *Starting with a 5-hole, Alice can obtain a 6-hole in the game within the next two rounds.*

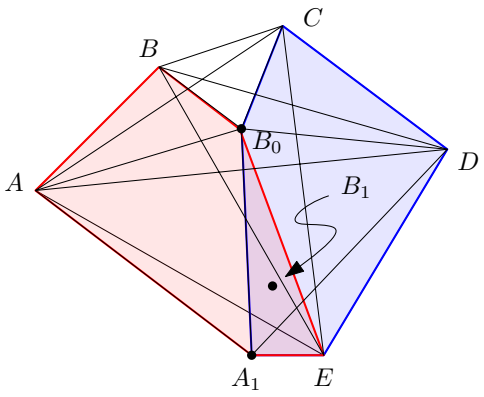


Figure 1: Formation of a 6-hole.

Proof. We note that there are 5 chords of a 5-hole such that each divides the 5-hole into a 4-hole and 3-hole. Bob has to place B_0 in the common intersection of all the 4-holes to prevent Alice from winning in the immediate next round. Since there are 5 such 3-holes Alice can find two 4-holes intersecting only in one edge, after the placement of B_0. One instance is depicted in Figure 1, where we start with the 5-hole $ABCDE$ and $\overline{B_0E}$ is the common edge between two 4-holes ABB_0E and $CDEB_0$. Alice places A_1 in such a way that it creates two 5-holes intersecting in one triangle ($\triangle B_0EA_1$ in the figure). If Bob places his point inside this triangle, then two 5-holes remain in the plane. Namely $ABB_0B_1A_1$ and EB_1B_0CD for the instance in Figure 1. Otherwise, Bob places his point inside only one of the two 5-holes. Thus after placement of B_1, Alice extends the remaining 5-hole to a 6-hole in the next round. □

Lemma 8 *Starting with a 6-hole, Alice can obtain a 7-hole in the game within the next two rounds.*

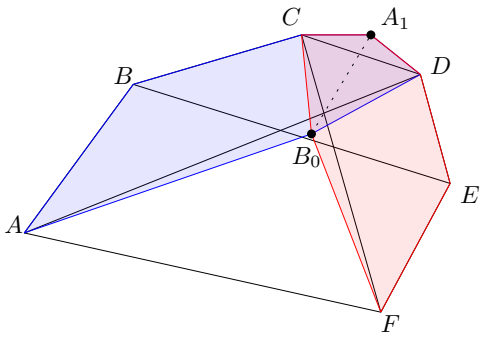


Figure 2: Formation of a 7-hole.

Proof. A 6-hole has three chords such that each of them divides the 6-hole into two 4-holes. An instance is depicted in Figure 2. $ABCDEF$ is the initial 6-hole and $\overline{AD}, \overline{BE}$, and $\overline{CF}$ are the chords dividing it into 4-holes. Since Bob can place B_0 inside the intersection of at most three out of these six 4-holes, Alice can find two 5-holes intersecting in one triangle after the placement of B_0 inside the 6-hole. She can extend these two 5-holes into two 6-holes intersecting in one convex quadrilateral. As a result after the placement of B_1 at least one 6-hole remains in the plane that can be extended to a 7-hole in the next round. Figure 2 depicts the case where B_0 is inside three 4-holes namely $ADEF$, $ABCF$ and $BCDE$. Thus, Alice creates two 6-holes $ABCA_1DB_0$ and CA_1DEFB_0 by placing A_1. Bob will try to place B_1 in the intersection of these two 6-holes formed after placement of A_1, but B_1 can be placed inside at most one of the two triangles $\triangle B_0CA_1$ or $\triangle B_0DA_1$. Even if Bob chooses one of them to place B_0 inside, either $B_1A_1DEFB_0$ or $ABCA_1B_1B_0$ remains a 6-hole ensuring the formation of a 7-hole in the next round by placing A_2 accordingly. The other cases arising from different placements of B_0 are analogous considering the symmetry of the chords of the initial 6-hole. □

Combining Lemma 6, 7, and 8 we get the following theorem.

Theorem 1 *Alice can win the ESMB game by obtaining a 7-hole from 15 points in the 8^{th} round.*

We note that in the cases of obtaining k-holes for $k = 6$ and 7, we considered the chords that divided the $(k-1)$-hole into half where the size of the half was $k-2$. As a result, after placement of B_0 at least two $(k-2)$-holes remain in the plane. That gives Alice a chance to create two $(k-1)$-holes intersecting in a triangle or a convex quadrilateral. This observation is not true when we start with a 7-hole and as a result, we can not ensure Alice's winning within the next 2 rounds. We prove Alice needs 5 more rounds to obtain an 8-hole starting with a 7-hole.

Lemma 9 *Starting with a 7-hole, Alice can obtain an 8-hole in the game within the next five rounds.*

Proof. We begin with a similar approach as Lemma 8 that there are 7 chords in a 7-hole such that each divides the hole into one 5-hole and one 4-hole. The chords are depicted in Figure 3 as $\overline{AD}, \overline{BE}, \overline{CF}, \overline{DG}, \overline{EA}, \overline{FB}$, and $\overline{GC}$ in the 7-hole $ABCDEFG$. To play optimally Bob places B_0 inside the 7-hole. After the placement of B_0, Alice can place A_1 in such a way that there are two 6-holes in the plane intersecting in one triangle. One instance is depicted in Figure 3 (left) with the two 6-holes namely AB_0EA_1FG and $BCDEA_1B_0$. If both the 6-holes remain after placement of B_1 Alice extends these two six holes into two 7-holes by a similar strategy in Lemma 8 and wins in the following round. Thus Bob must place B_1 inside at least one of these two 6-holes. Furthermore, Bob does not place B_1 inside the octagon $ABCDEA_1FG$ such

that five vertices of the octagon are lying on the same side of the line passing through B_0 and B_1, as this will create a 7-hole. Thus we have the following observation.

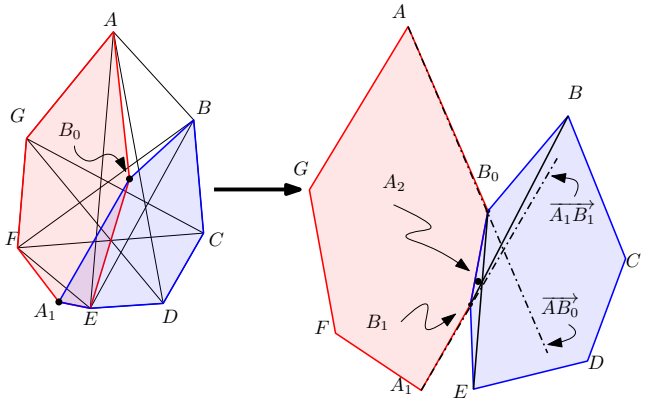


Figure 3: Observation 1.

Observation 1 *After the placement of B_1, there are two 6-holes in the plane sharing exactly one common edge. Alice can place A_2 in such a way that it creates two 6-holes and one 7-hole. Moreover, both the 6-holes share exactly one edge with the 7-hole and these two shared edges are adjacent to each other in the 7-hole.*

Precisely the position of A_2 as mentioned in Observation 1 is inside the intersection of the triangles $\triangle B_0BB_1$, $\triangle B_0EB_1$ and the triangle formed by the sides $\overline{B_0B_1}$, $\overrightarrow{AB_0}$ and $\overrightarrow{A_1B_1}$. Here $\overrightarrow{AB}$ denotes the prolongation of the segment $\overline{AB}$ from B. Now Bob must place B_2 inside the 7-hole to prevent Alice from winning in the immediate next round. Thus after placement of B_2, we have the following observation.

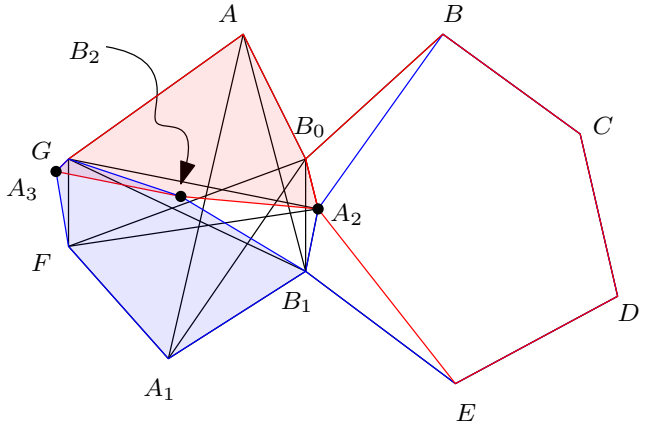


Figure 4: Observation 2.

Observation 2 *Alice can place A_3 in such a way that after the placement of B_3 there are two 6-holes in the plane sharing exactly one vertex.*

Proof of Observation. Note that Bob does not place B_2 in such a way that there are two 6-holes sharing only one vertex as this will help Alice to create two 7-holes sharing only one edge and win immediately. Moreover, there will be two 6-holes involving the vertices of the 7-hole (containing B_2 and B_3) following the same argument as Observation 1. If Alice can force the placement of B_3 such that the line passing through B_2 and B_3 intersects either $\overline{B_0A_2}$ or $\overline{A_2B_1}$ then Observation 2 holds. Thus to force such placement of B_3 Alice places A_3 in such a way that there are two 6-holes intersecting in one triangle such that none of them contain both $\overline{B_0A_2}$ and $\overline{A_2B_1}$ as their edges. one instance is depicted in Figure 4). △

Using Observation 2, Alice extends both the 6-holes to two disjoint seven holes by placing A_4 accordingly ensuring the formation of an 8-hole in the next round. The placement of A_4 for the instance considered in Figure 4 is depicted in Figure 5. □

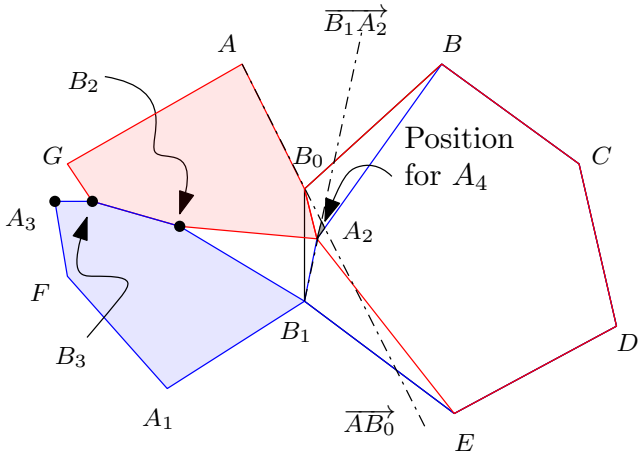


Figure 5: Placement of A_4 to obtain an 8-hole.

Thus by combining Theorem 1 and Lemma 9, we have the following theorem.

Theorem 2 *Alice can win the ESMB game by obtaining an 8-hole from 25 points in the 12^{th} round.*

2.1 Winning strategy for Maker in general with a higher speed

Now we present a general strategy for Alice to win the game when she benefits with a higher speed than Bob. First, we assume in each round Alice places 2 points while Bob places only 1. We show that Alice can obtain any k-hole for any given k in $2^{(k-1)}$ round.

Lemma 10 *Alice can win a Maker-Breaker version of the Erdos-Szekers game by obtaining a k-hole in $2^{(k-1)}$ round for any given k if she is allowed to place two*

points in each round while Bob is allowed to place only one point each round.

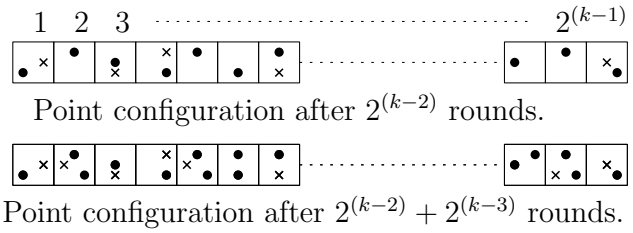


Figure 6: Game configurations when Alice has double speed than Bob.

Proof. We prove the lemma by describing the strategy of Alice. Consider $2^{(k-1)}$ disjoint unit squares in the plane. Alice places two points inside two different empty squares in each round for the first $2^{(k-2)}$ rounds. In the next $2^{(k-3)}$ rounds she places her points only in the squares where there are no points of Bob inside. She can always find $2^{(k-2)}$ squares since Bob can only place $2^{(k-2)}$ points in the first $2^{(k-2)}$ round. In the following $2^{(k-4)}$ round she places her points into the squares without any points of Bob. Following the strategy she can keep placing her points only in the squares without Bob's points and as a result, she can place them into a convex position achieving a k-hole in the $2^{(k-1)}$ round. □

Now we generalize the idea where the ratio of the points placed by Alice and Bob is $(1+\varepsilon)$ for any small $\varepsilon > 0$. In other words, Alice places at least one point more than Bob after r^{th} round of the game. Then using the similar argument of Lemma 10 we can conclude that after a finite number of steps, Alice can secure at least one such square that is free from a point of Bob. Thus she iterates the strategy to achieve one square containing only k points placed by her. This takes $(r+1)^{(k-1)}$ turns to win the game. Hence we have the following theorem.

Theorem 3 *Alice can win the ESMB game by obtaining a k-hole for any given positive integer k if the ratio of the points placed by Alice to the points placed by Bob is $(1+\varepsilon)$ for any $\varepsilon > 0$.*

3 ESMB game for equilateral holes

In this section, we address the question of obtaining equilateral holes for Maker. This question does not arise in the case of the classical Erdős-Szekeres problem: it is trivial to generate a point set of any size where no two pairs of points have the same distance between them.

We prove that Alice can create an equilateral 4-hole in the game. We describe the strategy in this subsection. It can be followed that Alice can create an equilateral triangle in the plane by placing A_2 in the second round and Bob must place her point inside the triangle to prevent Alice from winning in the immediate next round. Moreover, Bob ensures that B_2 is placed in such a way that it is not equidistant from two points in $\{A_1, B_1, A_2\}$. In the third round, Alice places A_3 dividing one outer angle ($\angle A_1B_1A_2$ in the Figure 7) of the triangle in such a way that the length of $\overline{A_3B_1}$ is same as the length of $\overline{A_1B_1}$. Moreover, the circle centered at A_3 with a radius of the same length as the length of $\overline{A_3B_1}$ intersects both the circles centered at A_1 and A_2 of the same radius. These two intersection points act as two potential candidates for A_4 such that either $A_1A_4A_3B_1$ or $A_2A_4A_3B_1$ becomes an equilateral 4-hole depending on the placement of B_3. This gives the following result.

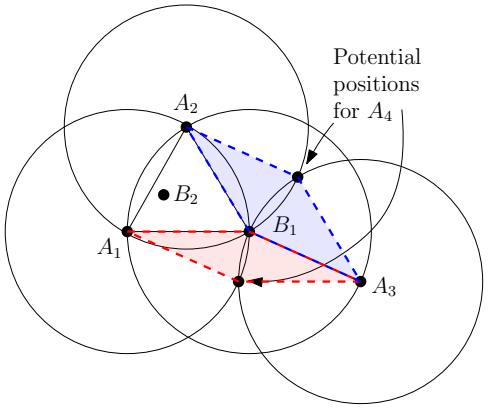


Figure 7: Formation of an equilateral 4-hole.

Theorem 4 *Alice wins the ESMB game for an equilateral 4-hole by obtaining it from 7 points in the 4^{th} round.*

4 ESMB game on a square grid

In this subsection, we study the game in a square grid of a fixed size, say $n \times n$. We show that if we allow the players to violate the general position assumption of the points, then Alice can win if and only if $n \geq \lceil \frac{k}{2} \rceil$. If the grid is of size $\lceil \frac{k}{2} \rceil$ it is easy to follow that Alice can ignore the placement of Bob and can form a k-hole from two consecutive rows or columns of the grid. But, interestingly, the bound is tight as Bob can prevent Alice from winning if $n < \lceil \frac{k}{2} \rceil$.

Lemma 11 *Alice does not have a winning strategy for the ESMB game on an $n \times n$ square grid if $n < \lceil \frac{k}{2} \rceil$ with $k > 2$.*

Proof. We note that Alice can not form a convex k hole only by using the points from two consecutive rows or columns. Hence she has to use points from at least

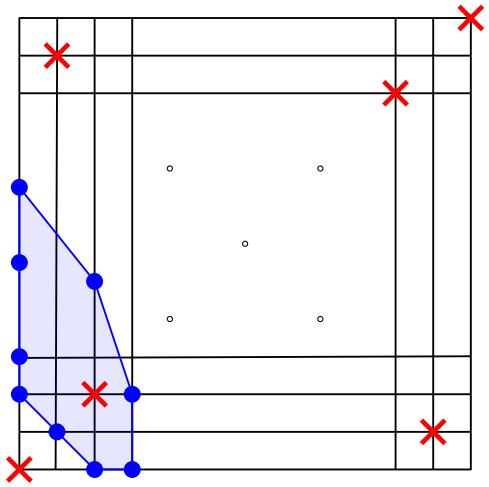


Figure 8: Winning strategy for Bob on a grid of $n \times n$ with $n < \lceil \frac{k}{2} \rceil$.

three rows or columns. Since any convex polygon of size k must contain two points of at least one diagonal of the grid, Bob places his n points alternatively on both the diagonals depending on the placements of the points by Alice, shown in Figure 8. This prohibits Alice from obtaining a hole of the desired size. □

Thus we have the following theorem.

Theorem 5 *Let us consider the ESMB game on an $n \times n$ square grid where both the players have to place their points on one of the vertices of the grid and they are allowed to place three or more colinear points. Alice can win the game by obtaining a k-hole for any positive integer $k > 2$, if and only if $n \geq \lceil \frac{k}{2} \rceil$.*

5 Conclusion

For Maker with a slightly higher speed than Breaker we have presented a general strategy to win the ESMB game, but it takes exponentially long to finish. Also, we observe that for small k like 8, the game finishes much faster. Thus it is an intriguing open question to address whether there exists a winning strategy for Maker with the same speed as Breaker even if there are constructions of the large sets without hole [4, 10, 13, 14]. An important observation is that the *Horton sets* [10], which are the building blocks of large point sets without k-holes for $k > 7$, are *fragile* in the sense that inserting one unwanted point in the set can create an unwanted hole. Moreover, the expected number of holes in a random point set of size n selected from a convex shape of unit area in the plane is $O(n^2)$ [2]. On the other hand the existing results on the expected size of the largest hole in random point sets [3] are logarithmic in terms of the number of points. Therefore, can Breaker delay the game infinitely by preventing Maker from forming a k-hole? If so, it is also interesting to study the maximum value of k for which Maker can always win.

References

[1] O. Aichholzer, J. M. Díaz-Báñez, T. Hackl, D. Orden, A. Pilz, I. Ventura, and B. Vogtenhuber. Erdős-szekeres-type games. In *Proc. 35^{th} European Workshop on Computational Geometry EuroCG'19*, pages 23–1, 2019.

[2] M. Balko, M. Scheucher, and P. Valtr. Holes and islands in random point sets. *Random Structures & Algorithms*, 60(3):308–326, 2022.

[3] J. Balogh, H. González-Aguilar, and G. Salazar. Large convex holes in random point sets. *Computational Geometry*, 46(6):725–733, 2013.

[4] D. Conlon and J. Lim. Fixing a hole. *Discrete & Computational Geometry*, 70(4):1551–1570, 2023.

[5] P. Erdős. Some more problems on elementary geometry. *Austral. Math. Soc. Gaz*, 5(2):52–54, 1978.

[6] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio mathematica*, 2:463–470, 1935.

[7] H. Harborth. Konvexe fünfecke in ebenen punktmengen. *Elemente der Mathematik*, 33:116–118, 1978.

[8] D. Hefetz, M. Krivelevich, M. Stojaković, and T. Szabó. *Positional games*, volume 44. Springer, 2014.

[9] M. J. Heule and M. Scheucher. Happy ending: An empty hexagon in every set of 30 points. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 61–80, 2024.

[10] J. D. Horton. Sets with no empty convex 7-gons. *Canadian Mathematical Bulletin*, 26(4):482–484, 1983.

[11] P. Kolipaka and S. Govindarajan. Two player game variant of the Erdös-Szekeres problem. *Discrete Mathematics & Theoretical Computer Science*, 15(Combinatorics), 2013.

[12] T. Valla. Ramsey theory and combinatorial games. *Master Thesis, Charles University in Prague, Czech Republic*, 2006.

[13] P. Valtr. Convex independent sets and 7-holes in restricted planar point sets. *Discrete & Computational Geometry*, 7:135–152, 1992.

[14] P. Valtr. Sets in $\mathbb{R}^d$ with no large empty convex subsets. *Discrete Mathematics*, 108(1-3):115–124, 1992.

The En Route Truck-Drone Delivery Problem

Danny Krizanc* Lata Narayanan† Jaroslav Opatrny‡ Denis Pankratov§

Abstract

We study the truck-drone cooperative delivery problem in a setting where a single truck carrying a drone travels at constant speed on a straight-line trajectory/street. The truck carries all items to be delivered. Delivery to clients located in the plane and not on the truck's trajectory is performed by the drone, which has limited carrying capacity and flying range, and whose battery can be recharged when on the truck. We show that the problem of maximizing the number of deliveries is strongly NP-hard even in this simple setting. We present a 2-approximation algorithm for the problem, and an optimal algorithm for a non-trivial family of instances.

1 Introduction

The use of unmanned aerial vehicles or drones for last-mile delivery in the logistics industry has received considerable attention in business and academic communities, see for example [1, 3, 15, 9]. Drones have been shown in a recent analysis [13] to have significantly less life-cycle costs, and faster delivery time compared to diesel or electric trucks in urban, suburban, and rural settings, and have less harmful emissions compared to diesel trucks. The potential applications where drone delivery could make a big impact include contactless delivery, return of unsatisfactory goods, rural or hard-to-access delivery and delivery in disaster relief scenarios.

In this paper we consider a system in which the delivery of physical items to clients located in the plane is done by *two cooperating mobile agents* having different but complementary properties. The first mobile agent, called the *drone* can move in any direction but it can travel only a limited distance, called its *flying range*, before it needs to recharge its battery. Furthermore, it has limited carrying capacity. The second mobile agent, called the *truck* can travel only along a fixed trajectory, called a *street* but its battery/fuel is not only sufficient to follow the street as long as necessary, but it is also equipped with a charging facility where the drone can recharge whenever it reaches the truck. Furthermore, it can carry all items that are to be delivered to the clients.

The delivery of items to clients is done as follows. All items to be delivered are preloaded on the truck at the warehouse. The truck then moves along the street at a fixed speed and it delivers items to any client who is located on its trajectory. The delivery of an item to a client who is not located on the trajectory of the truck must be carried out by the drone. At an appropriate time, the drone flies from the truck with the item to be delivered to the given client, drops the item there, and then flies back to the still-moving truck. There it can recharge, pick up another item, and make the next delivery, and so on. Clearly the same set-up can also be used to pick up items rather than deliver them. For ease of exposition, we always talk about item delivery in this paper.

Given a set of delivery locations and the parameters of the agents, i.e., the trajectory and the speed of the truck, the flying range of the drone and its speed, we want to compute a feasible schedule of deliveries that *maximizes the number of deliveries* made. Such a schedule specifies the order in which the deliveries to clients are done by the drone, and for each delivery it gives the time the drone leaves the truck. Clearly, to be feasible, the schedule should ensure that for each delivery, the drone can fly to the delivery location and back to the still-moving truck while having travelled distance at most its flying range, and arrive at the truck in time to start its next delivery.

1.1 Related work

The algorithmic study of truck-drone cooperative delivery problems was initiated by Murray and Chu [12] and Mathew et al. [11] where the problem of a single truck being helped by a single drone to deliver packages to customers is studied. Since then there has been a great deal of work (Murray and Chu's paper has received more than 1000 citations) on different versions of what is variously referred to as Truck-Drone Cooperative Delivery, Drone-Aided Delivery or Last-Mile Delivery problems. Variations considered include multiple trucks, multiple drones, drone-only delivery, mixed truck-drone delivery, etc. We refer the reader to recent surveys for more de-

*Department of Mathematics & Comp. Sci., Wesleyan University, Middletown, USA, dkrizanc@wesleyan.edu

†Department of CSSE, Concordia University, Montreal, Canada, lata.narayanan@concordia.ca

‡Department of CSSE, Concordia University, Montreal, Canada, opatrny@cs.concordia.ca

§Department of CSSE, Concordia University, Montreal, Canada, denis.pankratov@concordia.ca

tails [3, 4, 9, 15, 16].

In the above work, the problem is most often modelled using a weighted directed graph with customers as nodes, streets and drone flight paths as edges, etc. Under these circumstances the problems become versions of the Travelling Salesperson Problem or the Vehicle Routing Problem. As such they are all easily seen to be NP-hard in general and are solved by adapting known exact (e.g., Mixed Integer Linear Programming) or heuristic (e.g., greedy) techniques. For specialized domains some variants can be shown to be polynomial time, e.g. on trees [2].

In most of the previous research it is assumed that the points at which a truck and drone can rendezvous are part of the input (e.g., customer locations, depots) and that the truck or drone stops at the rendezvous point to wait for the other to arrive. More recent work [7, 8, 10, 14] has focused on the case where the rendezvous can occur "en route" as the truck is moving and the rendezvous points are to be determined by the algorithm, as is the case with our study. In these papers, the problems studied are again generalized versions of TSP or VRP and are attacked via adaptations of known exact or heuristic techniques. Here we restrict ourselves to the simplest version of the problem with one truck and one drone, where the truck travels at a constant speed along a single street. Surprisingly, even in this case, as shown in Section 3, the problem is strongly NP-hard.

All of the above work is concentrated on minimizing either the total delivery time or total energy requirements (or some combination of both) to deliver all of the packages to all of the customers. To the best of our knowledge we are the first to consider the problem of maximizing the number of clients that are satisfied in the en route model.

1.2 Our Truck-Drone Model

We define the truck-drone delivery problem more formally as follows. We assume that the delivery points as well as the trajectories of the truck and the drone, are set in the 2-dimensional Cartesian plane. Without loss of generality, we assume the warehouse is located at $[0, 0]$, and the truck starts fully loaded with all items to be delivered at the warehouse at time 0, and subsequently moves right on the x-axis with constant speed 1. Note that this allows us to measure the elapsed time by the distance of the truck from the origin.

The speed of the drone is denoted by v and it is assumed that v is a constant that is greater than 1. The *flying range* of the drone is given by the value R, and is defined as the maximum distance that the drone can fly on a full battery without needing to be recharged. We assume that the time to recharge the drone's battery, and to pick up an item from the truck, or to drop off an item at its delivery location are negligible compared to the delivery times, and thus are equal to 0. Therefore, any time the drone leaves the truck it can fly its full range R before returning to the truck.

We are given a multi-set $D = \{d_1, d_2, \ldots, d_n\}$ of delivery points in the plane where the deliveries are to be made. The truck delivers any item whose delivery point is located is on its trajectory, we assume that this can be done with negligible delay. Thus we assume below that none of the points in D is located on the trajectory of the truck, *i.e., on the positive x-axis.*

We now define a feasible delivery schedule for the truck-drone delivery problem.

Definition 1 *Given an instance $I = (v, R, D)$ of the truck-drone problem, where $D = \{d_1, d_2, \ldots, d_n\}$, we define a* schedule $\mathcal{S}_I$ *to be an* ordered list *of delivery points to which deliveries are made, and the start time of each delivery, i.e.,*

$$\mathcal{S}_I = ((d_{i_1}, s_1), (d_{i_2}, s_2), \ldots, (d_{i_m}, s_m)), m \leq n$$

where m is called the length *of the schedule and for $1 \leq j \leq m$ the drone makes a delivery to d_{i_j} by leaving the truck at point $[s_j, 0]$. The schedule is* feasible*, if $s_1 \geq 0$, and for each j, $1 \leq j \leq m-1$, the drone can reach d_{i_j} when leaving the truck at position $[s_j, 0]$ and return to the truck at or before $[s_{j+1}, 0]$.*

Schedule $\mathcal{S}_I$ is called *optimal* if there is no schedule that is longer than $\mathcal{S}_I$, that is, makes more deliveries than $\mathcal{S}_I$.

Given an instance $I = (v, R, D)$ of the truck-drone problem, where v and R are the speed and the range of the drone respectively, and D is the set of delivery points, the goal of the truck-drone delivery problem is to find an optimal delivery schedule.

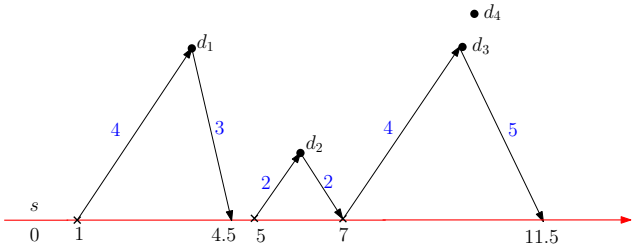


Figure 1: Instance $I = (2, 10, \{d_1, d_2, d_3, d_4\})$, and its schedule $\mathcal{S}_I = ((1, d_1), (5, d_2), (7, d_3))$. The trajectory of the drone is in blue, that of the truck in red. The blue numbers give the distances, the black numbers show the time sequence

.

Figure 1 shows an example of a truck-drone problem and of a feasible schedule.

1.3 Our results

In Section 3, we show that even for the ostensibly simple case of a single truck travelling on a straight line, and a single drone, the truck-drone delivery problem is strongly NP-hard. In particular, we show that given an instance I of the truck-drone problem and an integer k, it is strongly NP-hard [5] to decide whether there is a schedule $\mathcal{S}_I$ of length k.

In Section 4, we describe a greedy algorithm $\mathcal{A}_g$ and show that it computes a 2-approximation of an optimal schedule in $O(n^2)$ time. The factor of 2 is shown to be tight for this algorithm. Finally, in Section 5, we define a *proper* family of instances. Roughly speaking, in such instances, the delivery points do not have the same or "nearly" the same x-coordinates, where "nearly" depends on the difference in their y-coordinates. In particular, the greater the difference in the y-coordinates of the points, the greater is the difference in their x-coordinates in proper instances. We then give an $O(n^3)$ algorithm that calculates an optimal schedule for any proper instance.

Note that throughout this paper we assume that arithmetic operations, including taking square roots, can be done in constant time.

2 Preliminary Results

We say that a point $d = [x, y]$ is *reachable* by the drone from position $[s, 0]$ if the drone can leave the truck at $[s, 0]$, fly to point d and fly back to the truck with the total distance travelled at most its flying range R. First we examine some geometric properties of points in the plane that are reachable from $[s, 0]$ by the drone flying with speed v and having flying range R.

Suppose the drone leaves the truck at position $[s, 0]$, makes a delivery at $d = [x, y]$ and returns to the truck using its *full range* R. To fly range R the drone needs time $t = R/v$ and at that time the truck is at position $[s + R/v, 0]$. Therefore, the drone can make a delivery

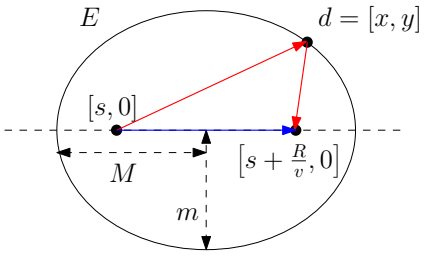


Figure 2: In Ellipse E shown above, the speed of the drone is not much higher than that of the truck. When the speed of the drone increases, the distance between the foci decreases, and the ellipse becomes closer to a circle.

at point $d = [x, y]$ if the total distance it flew satisfies the equation

$$|[s, 0], [x, y]| + |[x, y], [s + R/v, 0]| = R$$

Clearly all such points d reachable by the drone from $[s, 0]$ using its full flying range lie on *ellipse* E (see Figure 2) with left focus $[s, 0]$ and right focus $[s + R/v, 0]$. Furthermore, the *major radius*, i.e. the length of the *semi-major axis* of the ellipse is $M = \frac{R}{2}$, and *minor radius*, i.e. the length of its *semi-minor axis* is $m = \frac{R}{2v}\sqrt{v^2 - 1}$. Next, considering also the delivery points that can be reached by the drone by flying distance strictly less than R, we conclude that all points reachable from $[s, 0]$ by the drone *within* its flying range are located *on or inside* the ellipse E.

Assuming that the ellipse E is centered at $[0, 0]$, its left focus $[s, 0] = [-\frac{R}{2v}, 0]$, and its right focus is $[\frac{R}{2v}, 0]$, and M, m are the major, minor radii as specified above. The equation of the ellipse is:

$$\frac{x^2}{M^2} + \frac{y^2}{m^2} = 1 \tag{1}$$

Clearly, delivery to point $d = [x, y]$ is *feasible* only if $-m \leq y \leq m$, i.e., all delivery points should be located in a band of width $2m$ centered along the x -axis.

Assume a delivery point d is on the right half of ellipse E, and the drone makes a delivery to d starting from the truck at point $[s', 0]$ between the foci of the ellipse E. Since the distance from $[s', 0]$ to d is shorter than the distance from the left focus $[s, 0]$ of E to d, the drone can reach the delivery point d, flying for distance $< R$. However, the drone when leaving the truck at point $[s, 0]$ arrives at d *earlier* than when staying on the truck and leaving for d only later at point $[s', 0]$, and therefore it also returns to the truck earlier. Thus when using flying distance less than R the drone returns to the truck *later* as shown in Figure 3.

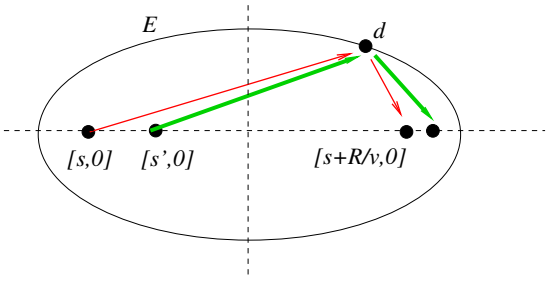


Figure 3: The red lines show the delivery with the full range R, the green lines show the delivery with range less than R.

This leads to the next lemma:

Lemma 1 *Consider a delivery point d in the right half of the ellipse E. To make a delivery to d flying less that the full range R, the drone must start the delivery at a*

point to the right of the left focus of E and the drone returns to the truck to the right of the right focus of E. Starting points for the drone to the left of the left focus are not feasible.

A symmetric observation holds about delivery points on the left half of E.

We now determine for each delivery point an interval on the trajectory of the truck describing feasible departure points for the drone to make a delivery to point d. Given a delivery point d, let E_1 and E_2 be the ellipses with major radius M and minor radius m, such that their foci are located on the x-axis, with E_1 containing d on its right half, while E_2 contains d on its left half. Let f_{i1}, f_{i2} be the foci of E_i for $i \in \{1, 2\}$ (see Figure 4). The following lemma now follows from Lemma 1 above.

Lemma 2 *Focus f_{11} is the point of the* earliest start *for a delivery to d, and focus f_{12} is the point of the* earliest return *to the truck from a delivery to d. Focus f_{21} is the point of the* latest start *for a delivery to d that can meet the truck, and Focus f_{22} is the* latest return *to the truck from any delivery to d. Feasible start points for delivery to d lie between f_{11} and f_{21}, with the corresponding return to the truck occurring between f_{12} and f_{22}.*

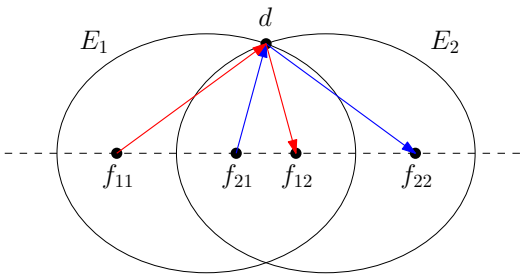


Figure 4: The red lines show the earliest delivery to d, the blue lines show the latest delivery to d. A delivery to d could be scheduled to start at a point between f_{11} and f_{21}.

In the rest of this paper, given a delivery point d we denote its earliest start time as $es(d)$ and the corresponding earliest return as $er(d)$, the latest start time of d as $ls(d)$, and the corresponding latest return back to the truck as $lr(d)$,

By the definition of the ellipse, for any delivery point d we have

$$er(d) - es(d) = lr(d) - ls(d) = R/v$$

the distance between the foci of E.

Given a point $d = [x, y]$, we can calculate the values $es(d)$, $ls(d)$ as follows. Imagine a horizontal line passing through d. It intersects the ellipse E centered at 0 at two points $[-x', y]$ and $[x', y]$. According to Equation (1), we have $(x')^2/M^2 + y^2/m = 1$. Therefore, $x' = M\sqrt{(1 - \frac{y^2}{m^2})}$. Now, imagine sliding the ellipse E along the x-axis. When E touches d for the first time, we obtain E_1 having travelled distance $x - x'$. Similarly, when E touches d for the last time, we obtain E_2 having travelled distance $x + x'$. Thus, we have:

Lemma 3 *For $d = [x, y]$*
$es(d) = x - \frac{R}{2v} - x'$, and $er(d) = es(d) + R/v$,
$ls(d) = x - \frac{R}{2v} + x'$, and $lr(d) = ls(d) + R/v$.

The next lemma gives the return point of the drone to the truck after a delivery to a delivery point $d = [x, y]$, starting from the truck at a position $[s, 0]$.

Lemma 4 *Suppose we wish to make a delivery to a delivery point $d = [x, y]$ using the drone, starting from the truck at position $[s, 0]$, and returning to the truck at position $[\mathrm{ret}, 0]$.*

1. *If $es(d) \le s \le ls(d)$,*

$$\mathrm{ret} = \mathrm{ret}(s, d, v) := s + \frac{s + av - x + z}{v^2 - 1} \qquad (2)$$

where $a = \sqrt{y^2 + (s - x)^2}$, $b = sv^2 + av - x$, and $z = \sqrt{b^2 - s(v^2 - 1)(b + s + av - x)}$

2. *If $s < es(d)$, then $\mathrm{ret}(s, d, v) = er(d)$.*

3. *If $s > ls(d)$, then delivery is impossible, thus we set $\mathrm{ret}(s, d, v) = \infty$.*

Proof. To see (1), observe that the total distance travelled by the drone is $d_1 = |[s, 0], [x, y]| + |[x, y], [\mathrm{ret}, 0]| = a + \sqrt{(\mathrm{ret} - x)^2 + y^2}$, which the drone travels in time d_1/v. At the same time the truck travels the distance $d_2 = \mathrm{ret} - s$. Thus we have the equation

$$\begin{aligned}
a + \sqrt{(x - \mathrm{ret})^2 + y^2} &= v(\mathrm{ret} - s) \\
(x - \mathrm{ret})^2 + y^2 &= (v(\mathrm{ret} - s) - a)^2 \\
x^2 - 2x\,\mathrm{ret} + \mathrm{ret}^2 + y^2 &= v^2(\mathrm{ret}^2 - 2s\,\mathrm{ret} + s^2) - \\
&\quad - 2av(\mathrm{ret} - s) + y^2 + \\
&\quad + (s^2 - 2sx + x^2) \\
\mathrm{ret}^2 - 2x\,\mathrm{ret} &= v^2\,\mathrm{ret}^2 - 2v^2 s\,\mathrm{ret} - 2av\,\mathrm{ret} + \\
&\quad + v^2 s^2 + 2avs + s^2 - 2sx
\end{aligned}$$

$$\begin{aligned}
0 &= (v^2 - 1)\,\mathrm{ret}^2 - 2(sv^2 + av - x)\,\mathrm{ret} + \\
&\quad + s(v^2 s + s + 2av - 2x) \\
0 &= (v^2 - 1)\,\mathrm{ret}^2 - 2b\,\mathrm{ret} + s(b + s + av - x),
\end{aligned}$$

and by solving the quadratic equation for ret $> s$ we have

$$\begin{aligned}\text{ret} &= \frac{b+\sqrt{b^2-s(v^2-1)(b+s+av-x)}}{v^2-1}\\ &= s+\frac{s+av-x+\sqrt{b^2-s(v^2-1)(b+s+av-x)}}{v^2-1},\end{aligned}$$

as needed.

For (2), note that if $s < es(d)$ then the drone remains on the truck until position $[es(d),0]$ is reached and then it starts a delivery from position $[es(d),0]$, since by Lemma 1, this gives the earliest time the drone can start from the truck for a delivery to point d. Thus for any such s the drone returns to the truck at position $[er(d),0]$.

Finally, (3) follows from Lemma 2. □

For s where $es(d) \le s \le ls(d)$ and a delivery point $d=[x,y]$, we call $\text{ret}(s,d,v)-s$ the *round-trip flight time* to d from $[s,0]$. It can be seen from Formula 2 that the round-trip flight time is not a linear function in s, which makes a calculation of a schedule for a given instance of the truck-drone problem more complicated. The following is a direct consequence of Lemma 1

Lemma 5 *For a delivery point $d=[x,y]$ and a point $[s,0]$ between $es(d)$ and $ls(d)$, the round-trip flight time $\text{ret}(s,d,v)-s$ reaches the maximal value R/v at $s=es(d)$, it decreases until $s=x(1-y/\sqrt{v^2-1})$ and then increases until $s=ls(d)$ where it again reaches the maximal value R/v.*

Lemma 6 *Let $d=[x,y]$ and $d'=[x',y']$ be two delivery points, and suppose there are valid drone trajectories from $[s,0]$ to d returning at $[r,0]$ and from $[s',0]$ to d' returning at $[r',0]$. If $s'<s<r\le r'$, then there is also a valid drone trajectory from $[s',0]$ to d returning at a point before $[r,0]$.*

Proof. Let R_1 be the length of the drone trajectory from $[s',0]$ to d' and then to $[r',0]$, and similarly, let R_2 be the length of the drone trajectory from $[s,0]$ to d and then to $[r,0]$. Then R_1/v and R_2/v respectively are the distances from $[s',0]$ to $[r',0]$ and from $[s,0]$ to $[r,0]$. Since $s'<s<r\le r'$, it follows that $R_2<R_1$. Now consider the ellipse E_1 with parameters (R_1,v) with $[s',0]$ as its left focus. Then d' is on the right half of E_1, and $[r',0]$ must be its right focus. Similarly, let E_2 be the ellipse with parameters (R_2,v) with $[s,0]$ and $[r,0]$ as its left and right foci respectively, and with d on the right half of the ellipse. Since $s'<s<r\le r'$, the ellipse E_2 is completely contained in E_1, and the point d is in the interior of the ellipse E_1. It follows that there is a valid drone trajectory to d starting at $[s',0]$. Furthermore, since the drone reaches d earlier if it starts at $[s',0]$ than if it stayed on the truck until $[s,0]$ and then flew to d, it must also return to the truck earlier than $[r,0]$. □

In the truck-drone instance that we use in the proof of strong NP-hardness in Section 3, many of the delivery points are located on the y axis. For these points we can simplify the expression used to define function $\text{ret}(s,d,v)-s$, and this simplified expression is used to obtain upper and lower bounds on $\text{ret}(s,d,v)-s$.

Lemma 7 *For $s\ge 0$ and a delivery point $d=[0,y]$ with $v/4\le y\le v/2$ we have*

$$\frac{2y}{v} < \text{ret}(s,d,v)-s < \frac{2y}{v}+\frac{1+4s^2+s}{v^2-1}.$$

Proof. Let $\Delta s := \text{ret}(s,d,v)-s$. Then the distance travelled by the drone is $\sqrt{s^2+y^2}+\sqrt{(s+\Delta s)^2+y^2}$. Since the drone travels at speed v, the time taken by the drone is then

$$\frac{\sqrt{s^2+2y^2}+\sqrt{(s+\Delta s)^2}}{v}.$$

During the delivery, the truck travels distance Δs at speed 1 taking the time Δs. Equating the two times we get:

$$\frac{\sqrt{s^2+y^2}+\sqrt{(s+\Delta s)^2+y^2}}{v}=\Delta s.$$

Solving for Δs, we obtain:

$$\Delta s=\frac{2(v\sqrt{s^2+y^2}+s)}{v^2-1}.$$

From this expression we immediately obtain the lower bound on Δs using $s\ge 0$:

$$\Delta s \ge \frac{2vy}{v^2-1}\ge\frac{2y}{v}.$$

Next observe that $\sqrt{s^2+y^2}\le y+\frac{s^2}{2y}$. Plugging this inequality into the expression for Δs we obtain:

$$\begin{aligned}\Delta s &\le \frac{2(v(y+s^2/2y)+s)}{v^2-1}=\frac{2vy+\frac{v}{y}s^2+2s}{v^2-1}=\\ &=\frac{2\left(1-\frac{1}{v^2}\right)vy+2\frac{1}{v^2}vy+\frac{v}{y}s^2+2s}{v^2-1}\\ &=\frac{2y}{v}+\frac{2\frac{y}{v}+\frac{v}{y}s^2+s}{v^2-1}\le\frac{2y}{v}+\frac{1+4s^2+s}{v^2-1},\end{aligned}$$

where in the last inequality we used the fact that $v/4\le y\le v/2$. □

3 Strong NP-hardness

In this section we prove that the following decision problem is strongly NP-hard:

Schedule Length problem. Given an instance I of the truck-drone problem, and an integer p, is there a schedule $\mathcal{S}_I$ of length p (that is, $\mathcal{S}_I$ makes p deliveries)?

We show below that there is a polynomial reduction from the well known 3-*Partition* problem [5] to the Schedule Length problem. Recall that in the 3-Partition problem we are given a multi-set of integers $Y = \{y_1 \leq y_2 \leq \cdots \leq y_n\}$, where $n = 3k$. Let $T = \sum_{i=1}^{n} y_i/k$. The 3-Partition problem asks if there is a partition of Y into k triples, such that the sum of elements in each triple is equal to T. The 3-Partition problem is strongly NP-hard [5].

Theorem 8 *The Schedule Length problem is strongly NP-hard.*

Proof. We prove the theorem by exhibiting a reduction from a 3-Partition instance $Y = \{y_1, y_2, \ldots, y_n\}$ to an instance I of the Schedule Length problem. We use the notation for the 3-Partition instance Y introduced immediately prior to the statement of the theorem. We assume that n is sufficiently large; the values in Y are bounded from above by a polynomial in n, so that $n^c < T \leq n^{c+1}$ for a sufficiently large constant c.

We now define the corresponding instance I of the Schedule Length problem as follows. The speed of the drone is set to $v = T$ and the flying range of the drone is set to $R = 4T$. Then the minor radius of the ellipse corresponding to the speed and range of the drone is $m = 2\sqrt{T^2 - 1}$.

For this proof, we depart from our convention of the truck starting at $[0, 0]$ and instead specify the starting position of the truck as $[2, 0]$ (this does not affect the complexity of the problem, but makes some of the formulas nicer). The set of delivery points D is partitioned into three subsets called A, B and C, that are defined below:
$A = \{[0, y_1], [0, y_2], \ldots, [0, y_n]\}$ is a set of delivery points located on the y-axis and correspond to the inputs to the 3-Partition problem.
$B = \{[6 + \epsilon(n), m], [2(6 + \epsilon(n)), m], \ldots, [(k - 1)(6 + \epsilon(n)), m]\}$ and
$C = \{[k(6 + \epsilon(n)), m], [k(6 + \epsilon(n)) + 4, m], \ldots, [k(6 + \epsilon(n)) + 4T, m]\}$
are sets of delivery points that are located at distance m from the x-axis and $\epsilon(n) \in (0, 1)$ is a function of n to be specified later.

Observe that each delivery point in $B \cup C$ can be reached by the drone from exactly one location on the x-axis, and the drone must fly its full range $R = 4T$ to make the delivery and return to the truck, and therefore, each such delivery takes time $R/v = 4$. See Figure 5 for an illustration of the instance I produced by the reduction, as well as the unique feasible drone trajectories for delivery points in B and C.

In total there are $n + (k - 1) + T + 1$ delivery points, and we set $p = n + (k - 1) + T + 1$ in the Schedule Length problem instance. In other words, this instance asks whether there is a schedule that delivers to *all* the delivery points. Observe that the number of points and their coordinates are all bounded by a polynomial in n, so the reduction runs in polynomial time.

We claim that the instance Y to the 3-Partition problem is a yes-instance if and only if I is a yes-instance to the Schedule Length problem.

It is clear that since the flying range of the drone equals $4T$, no deliveries to points in A can be scheduled after the deliveries to points in C are made. Thus a valid schedule delivering to all the points must schedule deliveries to A in the intervals between deliveries to points in B. There are k such intervals, and each interval is of length $2 + \epsilon(n)$. We claim that at most three points $[0, y_{i_1}], [0, y_{i_2}], [0, y_{i_3}]$ can be scheduled within such an interval and if only if $y_{i_1} + y_{i_2} + y_{i_3} \leq T$. Establishing this claim would finish the proof of the theorem.

Assume we have three integers $y_{i_1}, y_{i_2}, y_{i_3}$ such that $y_{i_1} + y_{i_2} + y_{i_3} \leq T$ and the truck with the drone on it is at position $[i(6 + \epsilon(n) + 2, m]$ for $0 \leq i \leq k - 1$. By the upper bound on the delivery time in Lemma 7 and observing that $i < k = n/3$, the total time for the three consecutive deliveries started at $[i(6+\epsilon(n)+2), m]$ is at most

$$\frac{2(y_{i_1} + y_{i_2} + y_{i_3})}{T} + 3\frac{1 + 4(k(6 + \epsilon(n)))^2 + (k(6 + \epsilon(n)))}{T^2 - 1}$$

$$\leq 2 + \frac{2n^2(6 + \epsilon(n))^2}{T^2 - 1} = 2 + O(n^2/T^2). \qquad (3)$$

Thus, the deliveries to $[0, y_{i_1}], [0, y_{i_2}], [0, y_{i_3}]$ can be completed before the delivery to $[(i + 1)(6 + \epsilon(n) + 2, m]$ is scheduled, provided that $O(n^2/T^2) = O(n^{-2c+2}) \leq \epsilon(n)$.

Assume we have three integers $y_{i_1}, y_{i_2}, y_{i_3}$ such that $y_{i_1} + y_{i_2} + y_{i_3} > T$ and the truck with the drone on it is at position $[i(6 + e(n) + 2, m]$ for $0 \leq i \leq k - 1$. By the lower bound on the delivery time in Lemma 7, the total time for the three consecutive deliveries started at $[i(6+\epsilon(n)+2, m]$ is at least $2(y_{i_1}+y_{i_2}+y_{i_3})/T > 2+1/T$ and they cannot be completed before the delivery to $[(i + 1)(6 + \epsilon(n) + 2, m]$ is scheduled, provided that the term $1/T \geq n^{-c-1}$ exceeds $\epsilon(n)$.

It is left to notice that because we can take n and c sufficiently large, we can find $\epsilon(n)$ satisfying:

$$O\left(\frac{1}{n^{2c-2}}\right) < \epsilon(n) < \frac{1}{n^{c+1}}.$$

For example, one could take $c = 4$ and $\epsilon(n) = 1/n^6$. This completes the proof of strong NP-hardness.

□

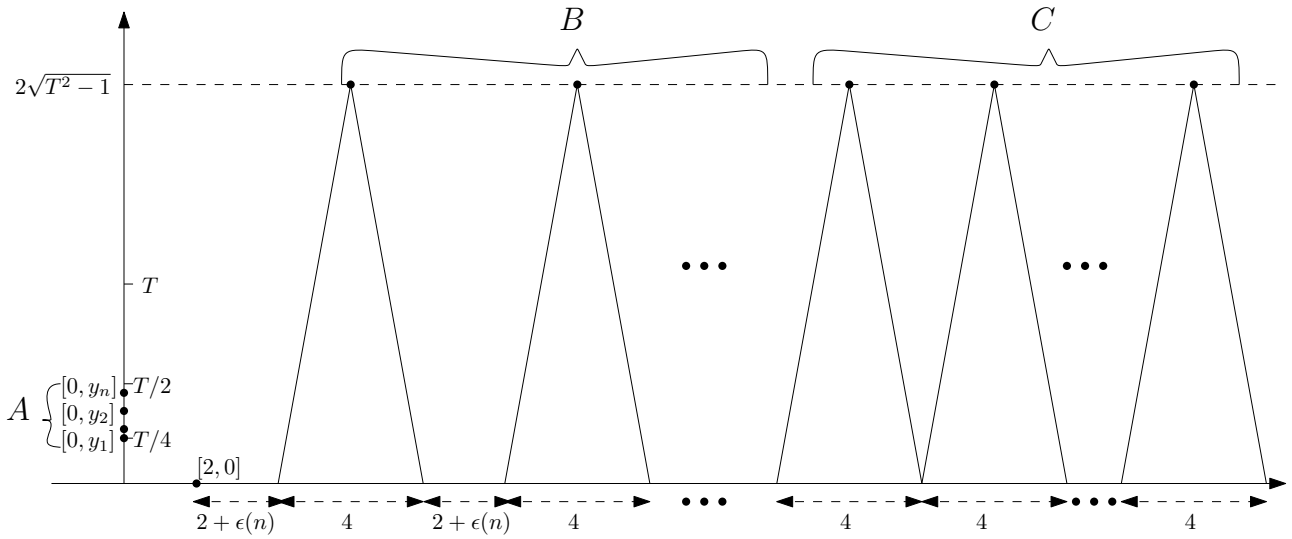


Figure 5: Illustration of the Schedule Length instance I output by the reduction from the 3-partition problem. The unique feasible drone trajectories for delivery points in B and C are also shown.

4 A Greedy Approximation Algorithm

In this section we describe a greedy scheduling algorithm for the truck-drone problem. Our algorithm, which we call $\mathcal{A}_g$, assigns deliveries to the drone as the truck moves from left to right starting from the initial position of the truck at $[0, 0]$. When the truck with the drone is at position $[s, 0]$, our greedy algorithm schedules a delivery to point d which, from among all feasible delivery points, minimizes the round-trip flight time from $[s, 0]$, i.e., which gives the *earliest possible return* for the drone to the truck. Notice that the delivery point which minimizes the round-trip flight time from $[s, 0]$ is not necessarily the delivery point that is at the shortest distance from $[s, 0]$. For example, in Figure 8, the point d_1 is closer than d_2 to $[s, 0]$. Thus one needs to use the function defined by Formula 2 to calculate which delivery point requires the shortest time to return to the truck. We then update s to be this shortest return time. If there are no feasible delivery points, then s i set to the earliest time any of the remaining points can be reached after the current time.

Algorithm 1 gives the pseudocode for $\mathcal{A}_g$. It is straightforward to see that Algorithm $\mathcal{A}_g$ can be implemented in $O(n^2)$ time, since a single evaluation of ret takes constant time. Figure 6 gives an example of the trajectories of the drone according to an optimal schedule and that of the schedule calculated by $\mathcal{A}_g$.

In the next theorem we compare the size of the schedule calculated by Algorithm $\mathcal{A}_g$ with respect to an optimal algorithm.

Theorem 9 *Given an instance $I = (v, R, D)$ of the truck-drone delivery problem, let $\mathcal{S}_g$ be the schedule produced by the algorithm $\mathcal{A}_g$ and let $\mathcal{S}_{OPT}$ be an optimal schedule. Then*

$$|\mathcal{S}_{OPT}| \leq 2|\mathcal{S}_g|.$$

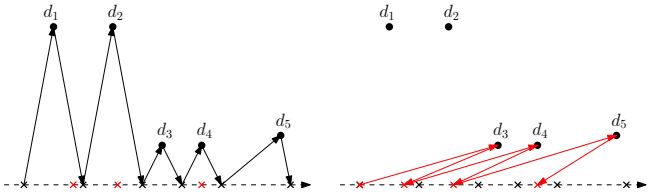


Figure 6: The black arrows, red arrows show the travel of the drone according to an optimal, greedy schedule, respectively, for an instance $I = (0, 4, 8, D)$ with D containing five delivery points $\{d_1, d_2, \ldots, d_5\}$. The black crosses and red crosses on the x-axis indicate the return points of OPT and $\mathcal{A}_g$, respectively.

Proof. Let $D = \{d_1, d_2, \ldots, d_n\}$ and let

$$\mathcal{S}_{OPT} = ((d_{i_1}, s_1), (d_{i_2}, s_2), \ldots, (d_{i_p}, s_p)), \text{ and}$$

$$\mathcal{S}_g = ((d_{j_1}, s'_1), (d_{j_2}, s'_2), \ldots, (d_{j_q}, s'_q)), \text{where } q \leq p \leq n.$$

We give a function $\mathcal{F}$ that maps delivery points in $\mathcal{S}_{OPT}$ to points in $\mathcal{S}_g$. For every k, with $1 \leq k \leq p$, define r_k to be the return time of the drone for the k^{th} delivery in S_{OPT} and similarly for every k, with $1 \leq k \leq q$, define r'_k to be the return time of the drone for the k^{th} delivery in $\mathcal{S}_g$. Define Q_k to be the set of delivery points in $\mathcal{S}_g$ whose return to the truck in the greedy schedule occurs *during* the flight time of the drone to deliver the k^{th} item in $\mathcal{S}_{OPT}$. That is,

$$Q_k = \{d_{j_\ell} : r'_\ell \in (s_k, r_k]\}$$

If $Q_k \neq \emptyset$, define $last(Q_k)$ to be the element of Q_k with the latest return according to the greedy schedule. Now define P_k to be the set of delivery points in $\mathcal{S}_g$ whose start time in the greedy schedule is before the start time of the k^{th} delivery in the optimal schedule, but whose return to the truck in the greedy schedule occurs between the return from the k^{th} delivery in the optimal schedule and the start of the $(k+1)^{st}$ delivery.

Algorithm 1 Greedy Approximation Algorithm $\mathcal{A}_g$ to Compute Feasible Delivery Schedule

Require: Instance $I = (v, R, D)$ where $D = \{d_1, d_2, \ldots, d_n\}$, is a list of delivery points.
Ensure: $\mathcal{S}_I$ is a feasible schedule of deliveries.

$\mathcal{S}_I \leftarrow L \leftarrow \emptyset$
$s \leftarrow 0$
▷ For each delivery point d_i, calculate $es(d_i)$ and $ls(d_i)$ and insert triple into L.
for $i = 1 \ldots n$ **do**
  if $s \leq ls(d_i)$ **then**
    $x.es = es(d_i)$
    $x.ls = ls(d_i)$
    $x.d = d_i$
    Insert(L, x)
  end if
end for
Sort$(L, key = es)$
while $L \neq \emptyset$ **do**
  $x \leftarrow first(L)$

  if $s < x.es$ **then**
    $s \leftarrow x.es$ ▷ If no feasible delivery point, move s forward.
  end if
  ▷ Find feasible delivery point which minimizes the return time to truck.
  $r_{min} \leftarrow \infty$
  while $x \neq NIL$ and $s \geq x.es$ **do**
    $r \leftarrow \text{ret}(s, v, x.d_i)$
    if $r < r_{min}$ **then**
      $r_{min} \leftarrow r$
      $save \leftarrow x$
    end if
    $x \leftarrow next(L)$
  end while
  ▷ Insert next delivery point into schedule, update s and list L
  Insert $(\mathcal{S}_I, (save.d, s))$
  $s \leftarrow r_{min}$
  for $x \in L$ **do**
    if $x.lr < s$ **then**
      Delete (L, x)
    end if
  end for
end while

That is:

$$P_k = \{d_{j_\ell} : s'_\ell \leq s_k \text{ and } r_k < r'_\ell \leq s_{k+1}\}$$

If $P_k \neq \emptyset$, note that it can have only one element, denote it as p_k.

We are now ready to define the function $\mathcal{F}$. For all $k \in \{1, \ldots, p\}$

$$\mathcal{F}(d_{i_k}) = \begin{cases} last(Q_k) & \text{if } Q_k \neq \emptyset \quad (4a) \\ p_k & \text{if } Q_k = \emptyset \text{ and } P_k \neq \emptyset \quad (4b) \\ d_{i_k} & \text{otherwise} \quad (4c) \end{cases}$$

We give an example to illustrate function $\mathcal{F}$ using an instance shown in Figure 6. In that case the optimal schedule makes 5 deliveries in order to $(d_1, d_2, d_3, d_4, d_5)$ and greedy schedule contains 3 deliveries (d_3, d_4, d_5) listed in order, omitting the starting times. For this case the function $\mathcal{F}$ is as follows:

$\mathcal{F}(d_1) = d_3$, $\mathcal{F}(d_2) = d_4$, $\mathcal{F}(d_3) = d_3$, $\mathcal{F}(d_5) = d_5$, and $\mathcal{F}(d_5) = d_5$.

First we prove that Clauses 4a, 4b, and 4c define a valid function on L', that is, every delivery point in the optimal schedule is mapped to a delivery point in the greedy schedule. Since Q_k and P_k only contain delivery points in the greedy schedule, the only case to consider is that $Q_k = P_k = \emptyset$ and $\mathcal{F}(d_{i_k}) = d_{i_k}$ and d_{i_k} is not part of the schedule $\mathcal{S}_g$ of the greedy algorithm $\mathcal{A}_g$.

Let ℓ be the largest integer such that $s'_\ell \leq s_k$. By assumption $d_{j_\ell} \neq d_{i_k}$. Since $Q_k = P_k = \emptyset$, either $r'_\ell > s_{k+1}$ or $r'_\ell \leq s_k$. If $r'_\ell \leq s_k$ (see Figure 7(a)), consider the $(\ell+1)^{st}$ delivery by the greedy algorithm. We know that $s'_{\ell+1} > s_k$ and since $Q_k = \emptyset$, it must be that $r'_{\ell+1} > r_k$. Thus for its $(\ell+1)^{st}$ delivery, the greedy heuristic should have chosen to deliver to d_{i_k} rather than to $d_{j_{\ell+1}}$, a contradiction.

Therefore it must be that $r'_\ell > s_{k+1}$. But then, using Lemma 6, there is a valid trajectory for the drone flying to d_{i_k} starting at s'_ℓ with an *earlier* return time that is at most $r_k \leq s_{k+1} < r'_\ell$ (see Figure 7(b)). Thus for its ℓ^{th} delivery, the greedy heuristic should have chosen to deliver to d_{i_k} rather than to d_{j_ℓ}, a contradiction. Thus d_{i_k} must be part of the greedy schedule, and $\mathcal{F}$ is a valid function mapping the delivery points in $\mathcal{S}_{OPT}$ to the delivery points in $\mathcal{S}_g$.

Finally, we claim that $\mathcal{F}$ maps at most two delivery points in $\mathcal{S}_{OPT}$ to one delivery point in $\mathcal{S}_g$. First, since the half-closed intervals $(s'_1, r'_1], (s'_2, r'_2], \ldots, (s'_{k'}, r'_{k'}]$ are all disjoint, and the half-closed intervals $(s_1, r_1], (s_2, r_2], \ldots, (s_{k'}, r_k]$ are also all disjoint, and any return point $[r', 0]$ can satisfy at most one of Clauses 4a and 4b, it follows that distinct elements in $\mathcal{S}_{OPT}$ are mapped to distinct elements of $\mathcal{S}_g$ by those two clauses. Second, clearly distinct elements in $\mathcal{S}_{OPT}$ are mapped to distinct elements of $\mathcal{S}_g$ by

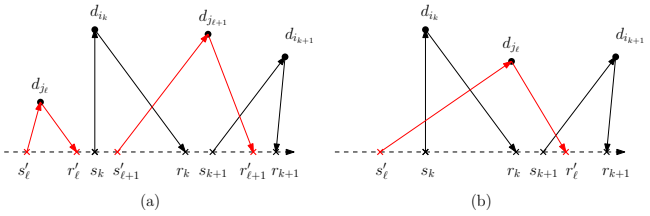


Figure 7: This figure illustrates the two cases in the proof that the function $\mathcal{F}$ in Theorem 9 is well defined. Red lines show deliveries in a presumed greedy schedule, and black lines indicate deliveries in a presumed optimal schedule.The dashed line represents the x axis, with distances from the origin marked.

Clause 4c. Therefore, the only kind of "collision" that can occur is that $\mathcal{F}(d_{i_k})$ is mapped to d_{j_ℓ} by Clause 4a or Clause 4b and $\mathcal{F}(d_{i_q})$ is mapped to d_{j_ℓ} by Clause 4c. This proves our claim that $\mathcal{F}$ maps at most two delivery points in $\mathcal{S}_{OPT}$ to one delivery point in $\mathcal{S}_g$.

We conclude that the schedule $\mathcal{S}_g$ created by $\mathcal{A}_g$ contains at least $\lceil p/2 \rceil$ elements, as desired. That is, $\mathcal{A}_g$ is a 2-approximation algorithm. □

The approximation ratio of 2 is tight. To see this, consider the instance given in Figure 8. For this instance the schedule computed by the greedy algorithm contains exactly one half of the delivery points, while an optimal schedule makes deliveries to all points. Thus, the approximation factor of 2 in Theorem 9 cannot be improved.

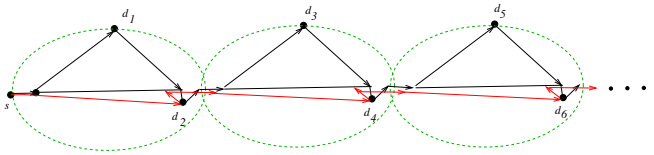


Figure 8: Approximation factor of 2 is sharp: in this instance, $d_1, d_3, d_5, \ldots$ are located at the maximal reach of the drone, and reachable only from the left focus of the corresponding ellipse (dashed green). An optimal schedule, shown in black, contains all points in order $d_1, d_2, d_3, d_4, \ldots$. The greedy algorithm, can immediately schedule a delivery to d_2, but not to d_1. After scheduling a delivery to d_2 a delivery to d_1 is not feasible any more, and this scheduling, shown in red, is repeated, resulting in the schedule $d_2, d_4, d_6, \ldots$.

5 Optimal algorithm for a restricted set of inputs

As seen in the proof of strong NP-hardness in Section 3, having many delivery points with the same x-coordinate creates a decision problem: should a delivery to a point $[0, y]$ be scheduled prior to or after the truck reaches $[0, 0]$. These decisions make the truck-drone problem NP-hard. In this section, we specify a family of instances called proper instances in which the delivery points do not have the same or "nearly" the same x-coordinates, where "nearly" depends on the difference in their y-coordinates. In particular, the greater the difference in the y-coordinates of the points, the greater is the difference in their x-coordinates in proper instances. We show that there is $O(n^3)$ algorithm to compute an optimal schedule for proper instances.

Definition 2 *Let $I = (v, R, D)$ be an instance of the truck-drone delivery problem where $D = \{d_1, d_2, \ldots, d_n\}$. We say I is a proper instance if:*

(1) for every $i, j \in \{1, \ldots, n\}$, with $i \neq j$, the delivery point d_j is not contained in the triangle $[es(d_i), 0], d_i, [lr(d_i), 0]$, and

(2) the set of closed intervals $\{[es(d_1), ls(d_1)], [es(d_2), ls(d_2)], \ldots, [es(d_n), ls(d_n)]\}$ form a proper interval graph [6], i.e., no interval in the set is a subset of another interval in the set.

Figure 9 shows an example of a proper instance. The definition of a proper instance implies that the delivery points have pairwise different x-coordinates and clearly, not many of them can reside in a narrow vertical band.

The lemma below implies that for a proper instance with $D = \{d_1, d_2, \ldots, d_n\}$, the intervals $[es(d_1), ls(d_1)], [es(d_2), ls(d_2)], \ldots, [es(d_n), ls(d_n)]$ are ordered by the x-coordinates of the corresponding points in D.

Lemma 10 *Let $I = (v, R, D)$ be a proper instance of the truck-drone delivery problem with $D = \{d_1, d_2, \ldots, d_n\}$. Let $d_i = [x_i, y_i]$ and $d_j = [x_j, y_j]$ be two points of D with $x_i < x_j$. Then either $ls(d_i) < es(d_j)$, or $es(d_i) < es(d_j) \leq ls(d_i) < ls(d_j)$*

Proof. If $y_j < y_i$ then $ls(d_j) > ls(d_i)$. Since interval $[es(d_j), ls(d_j)]$ cannot contain $[es(d_i), ls(d_i)]$, either $ls(d_i) < es(d_j)$, or $es(d_i) < es(d_j) \leq ls(d_i) < ls(d_j)$.

If $y_j > y_i$ then $es(d_j) > es(d_i)$. Since interval $[es(d_i), ls(d_i)]$ cannot contain $[es(d_j), ls(d_j)]$, either $ls(d_i) < es(d_j)$, or $es(d_i) < es(d_j) \leq ls(d_i) < ls(d_j)$. □

Given an instance $I = (v, R, D)$, we can verify if I is a proper instance in $O(n^2)$ time by checking each pair of intervals for non-containment, and each triangle for the non-inclusion of other points of D.

The following lemma is used to show that for proper instances we can restrict our attention to schedules in which the subsequent deliveries are ordered by the x -coordinates of delivery points, and in which the trajectories of the drone are non-crossing. See Figure 10 for an illustration.

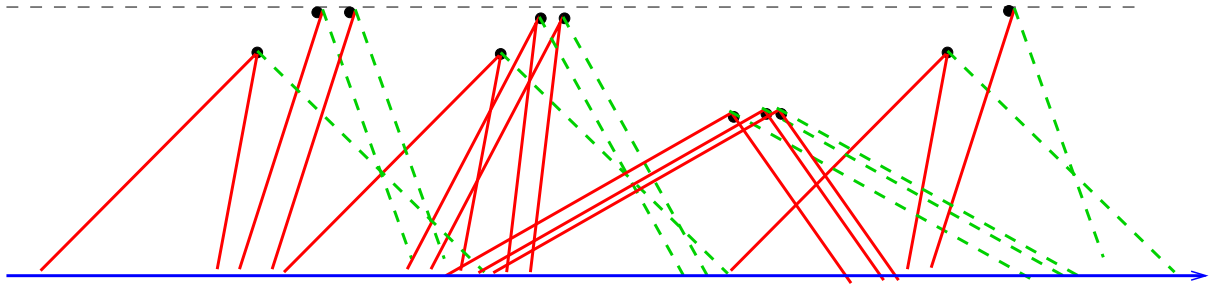


Figure 9: An example of a proper instance $I = (v, R, D)$ position for $v = 3$, and $r = 12$. For each delivery point the red segment points to the corresponding ed and ld points on the line, and the green segment points to the corresponding la. The three topmost points are at the limit of the reach of the drone.

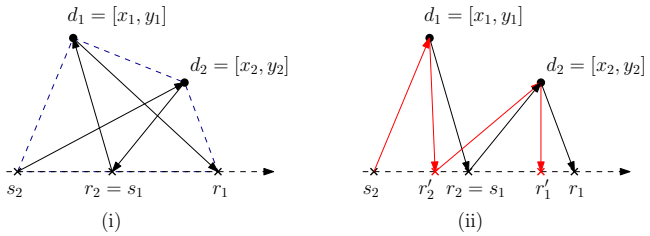


Figure 10: (i) The crossing trajectories to d_1 and d_2 are shown in black. (ii) The non-crossing trajectories, shorter in total, are in red.

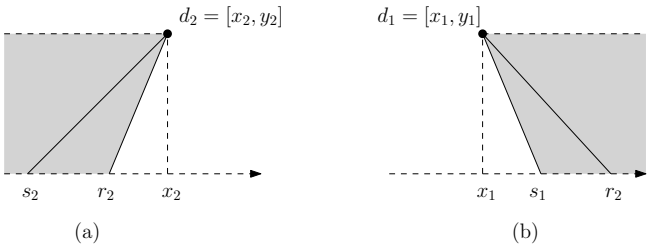


Figure 11: Illustration for the proof of (1) in Lemma 11. $d_1 = [x_1, y_1]$ and $d_2 = [x_2, y_2]$ with $x_1 < x_2$ and delivery to d_2 occurring before the delivery to d_1. Figure (a) illustrates the case of $y_1 \leq y_2$ and the shaded region demonstrates locations of d_1 which result in a crossing trajectory. Figure (b) illustrates the case of $y_1 \geq y_2$ and the shaded region demonstrates locations of d_2 which result in a crossing trajectory.

Lemma 11 *Let $I = (v, R, D)$ be a proper instance of the truck-drone problem. Assume that there is a feasible schedule for this instance in which a delivery to, say $d_2 = [x_2, y_2]$ immediately precedes that to $d_1 = [x_1, y_1]$, with $x_1 < x_2$. Then*

1. *The trajectories of the drone to d_1 and d_2 must cross.*
2. *By swapping the order of deliveries to d_1 and d_2 the total time of the two deliveries cannot increase, and thus swapping the two deliveries maintains the feasibility of the schedule, i.e., crossings of two consecutive trajectories can be avoided.*

Proof. To see (1), let s_i, r_i denote the start and return times to delivery point d_i for $i \in \{1, 2\}$. Assume for contradiction that delivery trajectories do not cross. If $y_1 \leq y_2$, then d_1 lies inside the triangle $[r_2, 0], d_2, [x_2, 0]$. This triangle is clearly contained in $[es(d_2), 0], d_2, [lr(d_2), 0]$ contradicting D being proper. If $y_1 \geq y_2$ then d_2 lies inside the triangle $[x_1, 0], d_1, [s_1, 0]$, which is contained inside $[es(d_1), 0], d_1, [lr(d_1), 0]$. This also contradicts D being proper. See Figure 11.

Next, we show (2). By Lemma 1 we can assume that the delivery to d_1 starts immediately at time r_2, i.e., $s_1 = r_2$ and terminates at time r_1. Clearly, in this case $es(d_2) < ls(d_1)$ and thus, by Lemma 10, $es(d_1) < es(d_2) \leq s_2 < r_2 \leq ls(d_1) < ls(d_2)$. Thus, a delivery to d_1 can be started at time s_2, and a delivery to d_2 can be started at time r_2 or later. It remains to show that the reversal in the delivery order can terminate latest at time r_1.

Suppose first that delivery to d_1, when started at time s_2 takes at most as much time as a delivery to d_2 at time s_2, see Figure 10 (ii). In the paragraph below, we use s_i to denote the point $[s_i, 0]$ and similarly r_i to denote the point $[r_i, 0]$. Consider the quadrilateral s_2,d_1,d_2,r_1 shown in blue. Since our instance is a proper instance, the triangle s_2,d_1,r_1 doesn't contain d_2 and thus this quadrilateral is convex. By the triangular inequality the sum $|s_2, d_1| + |d_2, r_1|$ of the lengths of two opposite sides of the quadrilateral is strictly less than the sum of the length of its diagonals $|d_1, r_1| + |s_2, d_2|$. Therefore,

$$|s_2, d_1| + |d_1, r_2| + |r_2, d_2| + |d_2, r_1| < \\ < |s_2, d_2| + |d_2, r_2| + |r_2, d_1| + |d_1, r_1|$$

and the path s_2, d_1, r_2, d_2, r_1 is shorter than the trajectory s_2, d_2, r_2, d_1, r_1. However, the path s_2, d_1, r_2, d_2, r_1 is *not necessarily a valid drone trajectory* if the delivery to d_1 from s_2 takes less time than the delivery to d_2

from s_2. Then, when delivering to d_1 first, the drone returns to the truck at point r'_2 located strictly between s_2 and r_2. But then the path s_2, d_1, r'_2, d_2, r_1 is even shorter than path s_2, d_1, r_2, d_2, r_1. Thus, when starting the delivery to d_2 at r'_2, the drone returns to the truck at a point r'_1 to the left of r_1, which improves the total delivery time to d_1 and d_2.

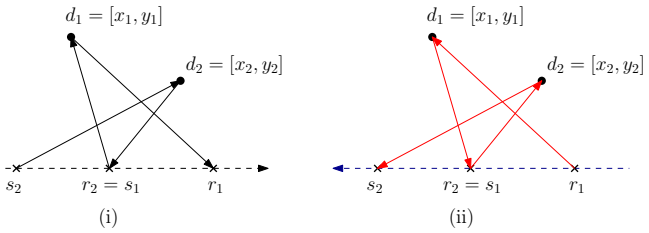


Figure 12: Reversing the directions of deliveries and the movement of the truck in (ii) converts a configuration of the second case to the first case

Now suppose instead that delivery to d_1, when started from s_2, takes more time then the delivery to d_2 from s_2, as for example on Figure 12 (i). By the shape of the function $\text{ret}(s, d, v))$, see Lemma 5, and since $es(d_1) < es(d_2)$ and $ls(d_1) < ls(d_2)$, a delivery to d_1, when started from s_1 also takes more time than the delivery to d_2 from s_1. Consider the configuration on Figure 12(ii) in which we reverse the movement of the drone and of the truck. Then we reduced this to the previous case and a delivery from s_1 first to d_2 and then to d_1 is shorter, and by reversing this once more we obtain that the delivery from s_2 first to d_1 and then to d_2 is shorter. □

A proper instance is guaranteed to have an optimal schedule with non-crossing trajectories. However not all optimal schedules give non-crossing trajectories. Indeed there are non-proper instances where crossing of trajectories is required in any optimal schedule as demonstrated in Figure 13.

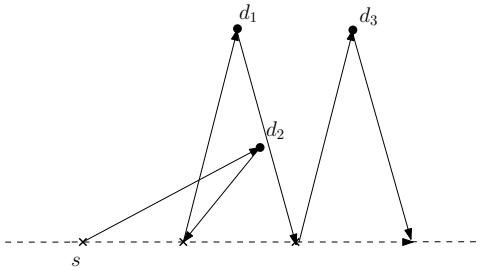


Figure 13: An instance of the problem where any optimal schedule must contain crossing trajectories. Points d_1 and d_3 are at the maximum reach of the drone. When scheduling a delivery first to d_1 then a delivery is possible either to d_2 or to d_3, but not to both.

Definition 3 *Let $I = (v, R, D)$ be an instance of the truck-drone delivery problem where $D = \{d_1, d_2, \ldots, d_n\}$. We call schedule*

$$\mathcal{S}_I = ((d_{i_1}, s_1), (d_{i_2}, s_2), \ldots, (d_{i_m}, s_m), m \leq n$$

monotone *if the x-coordinate of d_{i_j} is strictly less than the x-coordinate of $d_{i_{j+1}}$ for every $1 \leq j \leq m-1$,*

In the next theorem we show that there always exists a monotone schedule with the optimal substructure property for proper instances.

Theorem 12 *Let $I = (v, R, D)$ be a proper instance of the truck-drone delivery problem with $D = \{d_1, d_2, \ldots, d_n\}$. Assume that the points in D are listed according to increasing x-coordinate. Then there is an optimal schedule $\mathcal{S}_I = ((d_{i_1}, s_1), (d_{i_2}.s_2), \ldots, (d_{i_m}, s_m)$, $m \leq n$ for this instance with the following properties:*

1. *$\mathcal{S}_I$ is monotone.*
2. *For every $j \leq m$, the initial part $((d_{i_1}, s_1), (d_{i_2}, s_2), \ldots, (d_{i_j}, s_j))$ of $\mathcal{S}_I$ minimizes the delivery completion time for any subset of $\{d_1, d_2, \ldots, d_{i_j}\}$ of size j.*

Proof. Assume $I = (v, R, D)$ is a given proper instance of the truck-drone delivery problem and let $\mathcal{S}_I$ be an optimal schedule for it. By a repeated application of Lemma 11 we can swap any two consecutive deliveries that don't respect the order of x-coordinates of points, as in the bubble sort, while maintaining the schedule optimal. This eventually produces a monotone schedule of the same (optimal) length, proving (1).

To show (2), assume that for some $j \leq m$, there is a subset of j points $\{d_{i'_1}, d_{i'_2}, \ldots, d'_{i_j}\}$ of the set $\{d_1, d_2, \ldots, d_{i_j}\}$ for which there is a schedule $((d'_{i_1}, s'_1), (d'_{i_2}, s'_2), \ldots, (d'_{i_j}, s'_j))$ with $s'_j < s_j$ and which minimizes the delivery completion time for any subset of $\{d_1, d_2, \ldots, d_{i_j}\}$ of size j. Then by concatenating $((d'_{i_1}, s'_1), (d'_{i_2}, s'_2), \ldots, (d'_{i_j}, s'_j))$ with $((d_{i_{j+1}}, s_{j+1}), \ldots, (d_{i_m}, s_m)$, we get a valid schedule. In this manner, repeating the process starting with $j = i_m$ and decreasing appropriately the value of j we can get a schedule for I that is optimal, monotone, and satisfies the property 2 of the theorem. □

We use Theorem 12 to describe a dynamic programming algorithm that finds an optimal schedule for proper instances.

Theorem 13 *There is an $O(n^3)$ algorithm that calculates an optimal schedule for any proper instance $I = (v, R, D)$ of the truck-drone delivery problem.*

Proof. Assume the delivery points in D are listed in the order of their x coordinates. Define $T(i, j)$ to be

the earliest delivery completion time for the truck and the drone to perform exactly i deliveries from among $d_1, d_2, \ldots, d_j$ where d_j must be included in the schedule. If such a schedule is not possible, we define $T(i,j) = \infty$. We can compute $T(i,j)$ using dynamic programming as follows. We clearly have $T(1,j) = \text{ret}(s, d_j, v)$ for the base case of $i = 1$ (see Lemma 4 for the definition of ret) where $[s, 0]$ is the starting position of the truck. For $i \geq 2$, we have $T(i,j) = \min_{j'<j} \text{ret}(T(i-1,j'), d_j, v)$. This recursive formula immediately follows from the optimal substructure property stated in Theorem 12: a schedule resulting in the earliest completion time of making i out of the first j deliveries where d_j is included consists of delivering to $i-1$ out of the first $j' < j$ delivery points (with earliest completion time $T(i-1,j')$) followed by earliest delivery completion to d_j. Note that defining $\text{ret}(s', v, d) = \infty$ when $s' > ls(d)$ and $T(i,j) = \infty$ when delivery is impossible correctly works with the recursive computation of T.

Having computed T, we can find the maximum number of deliveries that can completed in a valid schedule by taking the maximum m such that $T(m,j) \neq \infty$ for some j. By recording for each (i,j) pair which choice of j' resulted in the table entry $T(i,j)$, we can reconstruct the schedule itself using standard backtracking techniques.

The running time is dominated by computing the table $T(i,j)$. It has $O(n^2)$ entries and each entry can be computed in time $O(n)$, since a single evaluation of ret takes constant time. The overall runtime is then $O(n^3)$. □

6 Discussion

We have shown that even in the simple case of a single drone with a single truck travelling in a straight line, the problem of coordinating their efforts to maximize the number of deliveries made is hard. Our work raises a number of different questions. We show that a greedy strategy achieves a 2-approximation. Is a better approximation possible? In particular, is the problem APX-hard or might there be a PTAS for it? Our implementation of the greedy strategy runs in $O(n^2)$ time. Is a better running time for the algorithm possible by taking advantage of the structure of the intervals created by the drone paths? The set of proper instances includes those where the y-coordinate is fixed. Could this be expanded to include points with a limited number of different y-coordinates? More generally, is there a "natural" setting in which the problem becomes fixed-parameter tractable? Finally, many variations on the problem are worth pursuing. Rather than maximizing the number of deliveries made with a given speed or drone range, one could consider the dual problems of minimizing the speed or range required to complete all deliveries. Versions with multiple drones and/or trucks, larger capacity drones, etc. are also of interest.

References

[1] A. Cornell, B. Kloss, and R. Riedel. Drones take to the sky, potentially disrupting last-mile delivery. *https://www.mckinsey.com/industries/aerospace-and-defense/our-insights/future-air-mobility-blog/drones-take-to-the-sky-potentially-disrupting-last-mile-delivery*, 2023.

[2] T. Erlebach, K. Luo, and F. C. R. Spieksma. Package delivery using drones with restricted movement areas. In S. W. Bae and H. Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 49:1–49:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[3] H. Eskandaripour and E. Boldsaikhan. Last-mile drone delivery: Past, present, and future. *Special Issue The Applications of Drones in Logistics*, 2023.

[4] J. C. Freitas, P. H. V. Penna, and T. A. Toffolo. Exact and heuristic approaches to truck–drone delivery problems. *EURO Journal on Transportation and Logistics*, 12:100094, 2023.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[6] M. C. Golumbic. Interval graphs and related topics. *Discrete Mathematics*, 55:113–121, 1985.

[7] A. Khanda, F. Corò, and S. K. Das. Drone-truck cooperated delivery under time varying dynamics. In *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, pages 24–29, 2022.

[8] H. Li, J. Chen, F. Wang, and Y. Zhao. Truck and drone routing problem with synchronization on arcs. *Naval Research Logistics (NRL)*, 69(6):884–901, 2022.

[9] G. Macrina, L. D. P. Pugliese, F. Guerriero, and G. Laporte. Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, 120:102762, 2020.

[10] A. Masone, S. Poikonen, and B. L. Golden. The multi-visit drone routing problem with edge launches: An iterative approach with discrete and continuous improvements. *Networks*, 80(2):193–215, 2022.

[11] N. Mathew, S. L. Smith, and S. L. Waslander. Optimal path planning in cooperative heterogeneous multi-robot delivery systems. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 407–423. Springer, 2015.

[12] C. C. Murray and A. G. Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

[13] A. Raghunatha, E. Lindkvist, P. Thollander, E. Hansson, and G. Jonsson. Critical assessment of emissions, costs, and time for last-mile goods delivery by drones versus trucks. *Scientific Reports*, 13(1):11814, 2023.

[14] T. Thomas, S. Srinivas, and C. Rajendran. Collaborative truck multi-drone delivery system considering drone scheduling and en route operations. *Annals of Operations Research*, pages 1–47, 2023.

[15] L. X., J. Tupayachi, A. Sharmin, and M. Ferguson. Drone-aided delivery methods, challenge, and the future: A methodological review. *Drones*, 7:191, 2023.

[16] R. Zhang, L. Dou, B. Xin, C. Chen, F. Deng, and J. Chen. A review on the truck and drone cooperative delivery problem. *Unmanned Systems*, pages 1–25, 2023.

Euclidean Freeze-Tag Problem on Plane*

Nicolas Bonichon† Cyril Gavoille† Nicolas Hanusse† Saeed Odak‡

Abstract

The freeze-tag problem is an optimization problem introduced by Arkin et al. (SODA'02). This problem revolves around efficiently waking up a swarm of inactive robots starting with a single active robot. Each asleep robot is activated by an awake robot going to its location. The objective is to minimize the total wake-up time for all robots, the *makespan*.

A recent paper by Bonichon et al. considers the geometric version of the freeze-tag problem on the plane. They conjectured that for the robots located on the plane with ℓ_2-norm, the makespan is at most $(1+2\sqrt{2})r$, where r is the maximum distance between the initial active robot and any asleep robot. In this paper, we prove the conjecture for the robots in the convex position and for $n \leq 7$ or $n \geq 281$, where n is the number of asleep robots (The conjecture was known to be true for $n \geq 528$ robots as shown by Bonichon et al.). Moreover, we show an upper bound of $4.63r$ for the makespan of robots in a disk of radius r in the ℓ_2-norm, improving the best known bound of $5\sqrt{2}r \approx 7.07r$.

1 Introduction

The *freeze-tag problem* is an optimization problem concerned with waking up a swarm of asleep (inactive) robots in the shortest possible time starting with a single awake (active) robot. Consider a set of robots represented by S and $|S| = n+1$ for $n \in \mathbb{N}$. Let $p_0, \ldots, p_n$ be the locations of the robots in a metric space, with p_0 being the location of the initial awake robot.

To activate an asleep robot, an awake robot must travel to the position of the asleep robot. As soon as an asleep robot is activated (awakened), it can assist in waking up the other asleep robots. We assume that each awake robot moves at the same speed of one unit per second while the asleep robots do not move. The *makespan* (*wake-up time*) is the time of the last wake-up. The freeze-tag problem has applications in group formation, searching, and recruitment in robotics, as well as broadcasting and IP multicast problems in network design (see [2, 9] and their references).

The problem can be rephrased as follows: A *wake-up tree* of S is a binary weighted spanning tree rooted at p_0 such that the degree of p_0 is one and the length of an edge is the distance between its endpoints (see for instance Figure 1). The freeze-tag problem is to find a wake-up tree of S with the minimum (weighted) depth.

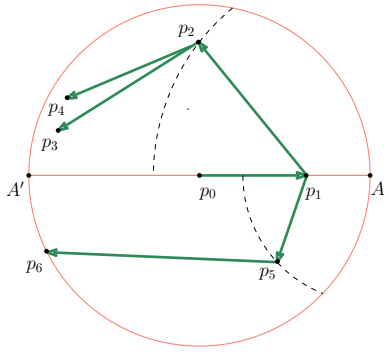


Figure 1: An example of a wake-up tree with 6 asleep robots in the Euclidean plane.

In this setup, Arkin et al. [3] give a constant approximation algorithm for the freeze-tag problem, when one asleep robot is located on each vertex. They also show that using an underlying graph metric, the problem is NP-hard. In a different paper, Arkin et al. [2] show that the freeze-tag problem, even on star metrics, is NP-complete. Moreover, they show that obtaining a 5/3-approximation is NP-hard for general metrics on weighted graphs. Therefore, a polynomial-time approximation scheme does not exist unless P = NP. In a related paper, Könemann et al. [9] consider the problem of finding a minimum diameter spanning tree with a bounded maximum degree in a complete undirected weighted graph and provide an $O(\sqrt{\log n})$-approximation algorithm for the freeze-tag problem in the general setting.

In this paper, we consider the *geometric freeze-tag problem* for the collection of robots. In the geometric freeze-tag problem, robots are modeled as points in $\mathbb{R}^d$ in a particular metric for some $d \in \mathbb{N}$. For $d = 3$ and ℓ_p norm, it has been shown that the geometric freeze-tag problem is NP-hard where $p \geq 1$ [6, 7, 10]. Sztainberg et al. [11] give a heuristic algorithm with a tight

*Due to the space constraint several proofs are omitted. For the proofs see the full version of the paper.

†LaBRI, University of Bordeaux, CNRS, Bordeaux INP, France

‡School of Electrical Engineering and Computer Science, University of Ottawa, Canada.

approximation of $\Theta(\log^{1-1/d} n)$ for the makespan of n asleep robots in d dimensional space. In particular, their greedy algorithm yields an $O(1)$-approximation in one dimension ($d = 1$) and an $O(\sqrt{\log n})$-approximation in two dimensions ($d = 2$). Arkin et al. in [2], for any constant $d \in \mathbb{N}$, provide a polynomial-time approximation scheme when robots are located in $\mathbb{R}^d$ equipped with ℓ_p metric. Moreover, their algorithm runs in time $O(n \log n + 2^{poly(1/\epsilon)})$.

It is worth mentioning that Hammar et al. [8] study the online freeze-tag where each asleep robot is revealed at a specified time. Later, an optimal algorithm for the online freeze-tag problem was introduced by Burnner et al. [5].

In the geometric setting, as long as normed spaces are concerned, the positions of all the robots can be scaled and translated such that all asleep robots are in a unit disk, and the initial active robot is at the origin (i.e., the active robot is at the origin and the distance between the active robot and the farthest asleep robot is a unit). Note that in this configuration, the makespan is always lower bounded by the maximal distance between the active robot and asleep robots (the radius of the unit disk). Combinatorial upper bounds for the makespan of robots in a unit disk with respect to ℓ_p norm are studied by Bonichon et al. [4]. In particular, when robots are located in the unit disk in the plane with ℓ_1 norm, they provide a tight strategy with makespan 5. They also show [4, Theorem 2] that the makespan for n asleep robots in the unit disk with one active robot at the origin is at most $3 + c/\sqrt{n}$, where c is a constant depending on the norm.

We focus on the *Euclidean freeze-tag problem on the plane*. That is, we consider robots as points in the Euclidean metric space on $\mathbb{R}^2$. In this setting, the problem remains NP-hard [1], and Najafi Yazdi et al. [12] provide an algorithm with a makespan $(5 + 2\sqrt{2} + \sqrt{5})$ for the robots located in a unit square that runs in linear time. Recently, Bonichon et al. [4] proposed an algorithm with a makespan $5\sqrt{2}$. They also conjectured that the maximum makespan of robots in a unit disk of any norm is achieved when the number of robots is four. For $n = 4$, we get the worst-case whenever four asleep robots p_1, p_2, p_3, and p_4 form a square with sides of length $\sqrt{2}$. It takes time 1 to go from the active robot p_0 in the center to p_1, and then one robot has to wake up p_2 followed by p_3 in time $2\sqrt{2}$, and the other one wakes up p_4. In the Euclidean freeze-tag problem, this conjecture translates to the following.

Conjecture 1 ([4]) *Let n be a positive integer. There exists a strategy to wake up n asleep robots inside a unit disk in Euclidean space starting with an active robot at the origin in time at most $1 + 2\sqrt{2}$.*

Our main contributions are the following. First we show that the Conjecture 1 holds for $n \leq 7$ or $n \geq 281$, and also when the robots are in convex position (Theorems 1, 2, and 3). Then we provide a new upper bound of 4.63 for the makespan of the Euclidean freeze-tag problem on the plane, improving upon the best-known result of $5\sqrt{2} \approx 7.07$. This also shows that the optimal upper bound for the Euclidean case is strictly less than the lower bound of 5 for the ℓ_1 norm.

The rest of this paper is organized as follows: The next section will be dedicated to preliminaries and some definitions for the geometric objects needed in the sequel. In Section 3, we discuss monotonic wake-up strategies for two simple geometric objects as a subroutine. In Section 4, as a warm-up, we prove Conjecture 1 for small values of n. Section 5 and Section 6 study the correctness of Conjecture 1 when the asleep robots are in a convex position and when the number of asleep robots is at least 281, respectively. Section 7 establishes an improved makespan of 4.63 for the Euclidean freeze-tag problem in the plane.

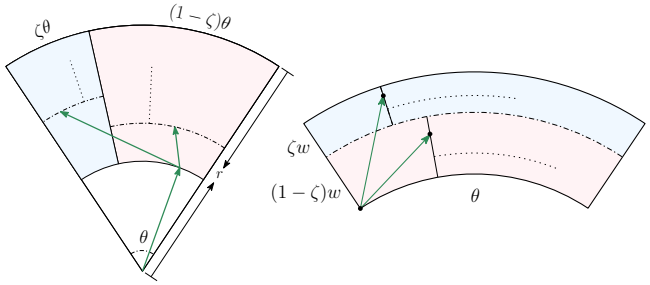
2 Preliminaries

For each $0 \leq i \leq n$, the *wake-up time of robot* p_i is the length of the path of p_i from p_0 in the wake-up tree. The depth of a wake-up tree indicates its makespan. Using this terminology, a closed geometric region $\mathcal{R}$ on the plane with a specified active robot has a makespan of at most τ if for every $n \in \mathbb{N}$, there exists a wake-up tree for every configuration of n asleep robots in the region $\mathcal{R}$ with a depth at most τ.

Many of the strategies that we will define rely on a recursive decomposition of regions R into subregions. Therefore, we will define some regions that will be useful to us later on.

A *cone* of angle θ and radius r is a geometric region inside a disk of radius r between two segments with one endpoint on the center of the disk and the other endpoint on the boundary such that the angle between the two segments is θ (see Figure 2(a)). The center of the disk is referred to as the cone's apex. A cone defined using a disk of radius one is called a *unit cone*.

A (unit) *crown* of angle θ and width w is obtained from a unit cone of angle θ by subtracting a smaller cone of the same angle and radius $1 - w$ (see Figure 2(b)). Each non-trivial crown consists of 4 sides: two curved sides and two straight-line sides. We call the longer curved side of a crown the exterior side and the shorter curved side of a crown is called the interior side. For future reference, we represent the makespan of a unit crown of angle θ and width w starting with two active robots at a corner on the exterior side of the crown with



(a) (b)

Figure 2: (a) A cone of angle θ and radius r. The monotonic wake-up strategy to solve a cone with one active robot in the apex. (b) A crown of angle θ and width w. The monotonic wake-up strategy to solve a crown with two active robots on a corner.

crown(w, θ).

A *dome* is the part of a cone between its arc and its chord (see Figure 6). The radius and angle of a dome are the same as the corresponding cone.

We define *semi-cone* to be the region enclosed between two chords of a unit disk with one common endpoint that does not contain the origin in its interior (see Figure 7(a)). We call the common endpoint of two chords the apex of the semi-cone.

Throughout this paper, ϕ stands for the golden ratio, i.e., $\phi = \frac{1+\sqrt{5}}{2}$. Note that $\phi^2 = \phi + 1$.

3 First Bounds For Geometric Shapes

In this section, we present *monotonic* wake-up strategies for two simpler geometric objects, namely the unit cone and the unit crown, as subroutines for the other algorithms discussed in the rest of the paper. We first assume that for cones (resp. crowns) robots are ordered w.r.t the distance (resp. angular distance) from the apex (resp. a corner). Given a binary wake-up tree, a strategy is said *monotonic* if for every path from the root to leaves, points are ordered w.r.t to the distance from the root.

We begin this section with a simple observation stating the triangle inequality in polar notation.

Observation 1 *Let $A = (r_a, \theta_a)$ and $B = (r_b, \theta_b)$ be two points in polar notation inside a unit disk. Then we have $||AB|| \leq |r_a - r_b| + \max(r_a, r_b) \cdot |\theta_a - \theta_b|$. In particular, since $r_a, r_b \leq 1$, we have $||AB|| \leq |r_a - r_b| + |\theta_a - \theta_b|$.*

In the following, $|r_a - r_b|$ and $\max(r_a, r_b) \cdot |\theta_a - \theta_b|$ are referred to as the radial distance and angular distance between A and B, respectively. As the first classical result, we present a result from [4] that establishes an upper bound for the makespan of robots positioned within a unit cone of angle θ (refer to Figure 2(a)).

Informally, the initial robot finds the closest asleep robot in the cone and the cone is partitioned into two regions, namely, subcones of angle $\zeta\phi$ and $(1-\zeta)\phi$. Then the process is repeated similarly in each region with one active robot. From Observation 1, the length of the path $p_0, p_1, \ldots p_\ell$ from the initial robot toward any other robot is at most $\sum_i |r_i - r_{i-1}| + |\theta_{p_i} - \theta_{p_{i-1}}|$. The sum of the first terms is at most 1 whereas for the angular detour we have $A(\theta) \leq \max\{\theta + A(\zeta\theta),\ (1-\zeta)\theta + A((1-\zeta)\theta)\}$. If we take $\zeta = 1/2, A(\theta) \leq \sum_{i\geq 0} \theta/2^i < 2\theta$. In fact, taking $\zeta = 2 - \phi$, we get:

Lemma 1 ([4](Proposition 14)) *There exists a strategy to wake up asleep robots in a cone of angle θ and radius one starting with one awake robot at the center of the cone in time at most $1 + \phi\theta$.*

As the next geometric subroutine, similar to Lemma 1, we can construct a monotonic wake-up strategy for a unit crown using a monotonic recursive partition into sub-crowns.

Lemma 2 *There exists a strategy to wake up all of the robots in a crown of angle θ and width w starting with two awake robots at a corner in time at most $\theta + \phi w$.*

If we consider only one awake robot on the boundary, we must consider an extra time to wake up another robot, and then we can apply the result of Lemma 2.

Corollary 1 *There exists a strategy to wake up all of the robots in a crown of angle θ and width w starting with one awake robot at a corner in time at most $\theta + (1+\phi)w$.*

Finally, with a strategy analogous to that in Lemma 2, one can wake up robots within a rectangular region.

Corollary 2 *There exists a strategy to wake up all of the robots in a rectangle of width w and height h starting with two awake robots at a corner in time at most $h + \phi w$.*

4 Configurations With Small Number of Asleep Robots

In this section, we state the correctness of Conjecture 1 for the small number of asleep robots.

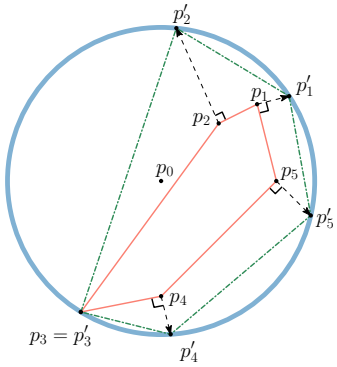


Figure 3: Projection of convex points on the disk.

Theorem 1 *Let $n \leq 7$ be a positive integer. There exists a strategy to wake up n robots in a unit disk in time less than $1 + 2\sqrt{2}$.*

The statement is trivial for $n \leq 3$. Let $n \in \{4, 5, 6, 7\}$ and p_1 be one of the robots such that the line passing through p_0 and p_1 cuts the unit disk C into two parts, each of which contains at most $\lceil (n-1)/2 \rceil$ robots (different from p_0 and p_1). Let A and A' be the intersection of this line with C, such that p_1 is on the segment p_0A.

The wake-up strategy is as follows: once the robot p_1 is awakened, the two robots are positioned at point A. Then, each robot takes care of half of the disk. In each half-disk, the robot will awaken the robot closest to A, and then the two robots will each awaken at most one more robot (see Figure 1).

5 Robots In Convex Position

In this section, we assume that the coordinates p_i are ordered in the counter-clockwise cyclic ordering. We present the following theorem:

Theorem 2 *If the point set corresponding to S is in a convex position within a disk C of radius one, the makespan of S is upper bounded by $1 + 2\sqrt{2}$.*

Let us sketch the proof of Theorem 2:

- For each $p_i \in S$, we assign the point $p'_i \in C$ being the intersection of the ray perpendicular to $p_{i-1}p_i$ emanating from p_i and going outside from the convex hull of S. This projection is such that for any pair p_i and p_j, $||p'_ip'_j|| \geq ||p_ip_j||$ (see Figure 3).
- If S' is a point set on the disk C, we provide a wake-up tree T' of makespan less or equal to $1 + 2\sqrt{2}$.
- The wake-up tree T on S is defined from T'. If (p'_i, p'_j) belongs to T' then (p_i, p_j) belongs to T. We show that the makespan of T is smaller or equal to the one of T' and thus bounded by $1 + 2\sqrt{2}$.

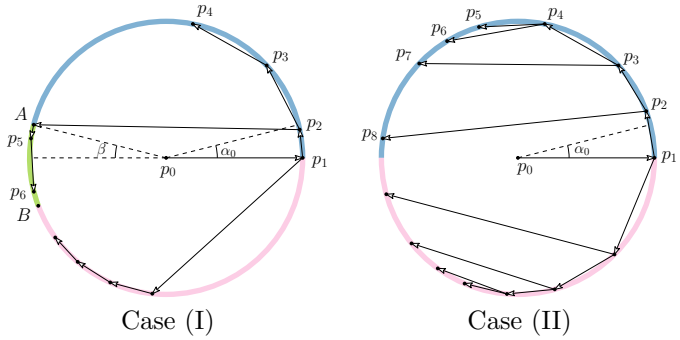
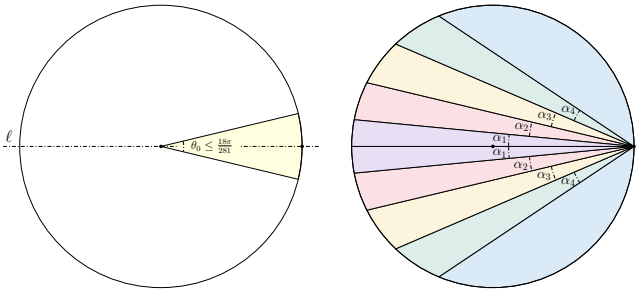


Figure 4: Robots on the disk. Case (I): The disk is partitioned into 3 arcs. Case (II): Without the existence of a small angle, the disk is partitioned into 2 half-disks. Except for the last robot p_8, robots p_2 to p_7 are awake using a monotonic complete binary tree from p_1.

Let's introduce a simple strategy to wake up an arc of angle α containing k asleep robots while the first robot p_j on the arc in the counter-clockwise order is awake. In the algorithm Arc Strategy, p_j wakes up p_{j+1}, and any *monotonic* complete binary wake-up tree can be considered. By monotonic, we mean that each robot wakes up another robot with a larger angular position. The depth of this binary wake-up tree is $1 + \lfloor \log_2 k \rfloor$.

Disk Strategy takes as an input the point set S on the boundary of a unit disk and an angle α_0. Take $\beta = \pi - 2\sqrt{2}$ and let $1 \leq i \leq n$ be an integer where the angle $\angle p_ip_0p_{i+1}$ is the smallest. We consider two cases (see Figure 4):

- Case (I): If $\angle p_ip_0p_{i+1} \leq \alpha_0$ then p_0 wakes up p_i. For convenience, assume that the line passing through p_0 and p_i is horizontal. The two robots located at p_i, wake up in parallel, all the robots at an angular distance at most $\pi - \beta$ from p_i; one going in the counter-clockwise order through p_{i+1} and the other one in the clockwise order through p_{i-1}. As soon as p_{i+1} is awake, it directly goes to the position of the disk at an angular distance $\pi - \beta$ from p_i and wakes up all the remaining robots in the remaining arc of angle 2β.
- Case (II): If the previous case does not hold, take an integer $1 \leq j \leq n$ such that the angle $\angle p_{j-1}p_0p_{j+1}$ is the smallest. Again, for convenience, assume that robots p_0 and p_j are located on a horizontal line. As in the previous case, p_0 wakes up p_j and robots on the two arcs up to the angular distance $\pi - \alpha_0$ from p_j are awake by two robots emanating from p_j using Arc Strategy. One robot from p_{j+1}



(a) (b)

Figure 5: Illustration of proof of Theorem 3. (a) stage one. (b) stage two: split into 8 semi-cones and 2 domes.

(resp. p_{j-1}) directly goes to the last robot on the arc $(\pi - \alpha_0, \pi)$ (resp. $(\alpha_0 - \pi, -\pi)$). Then the rest of the robots on the arcs of angle $\pi - \alpha_0$ are awake.

To conclude, we show that for $\alpha_0 = \pi/11$, the DISK STRATEGY has a makespan less than or equal to $1+2\sqrt{2}$.

6 Strategies For Large Number of Robots

By examining the constants closely, [4, Theorem 2] implies that Conjecture 1 is correct when n, the number of asleep robots, is at least 528. In line with [4, Theorem 2], we prove that Conjecture 1 holds when the number of asleep robots in the unit disk is at least 281. To get this lower bound on the number of robots, we need to introduce a wake-up strategy for the domes and semi-cones.

The sketch of our strategy is the following: (1) Since $n \geq 281$, there is a cone of the angle at most $\frac{18\pi}{281}$ such that it contains at least 9 asleep robots. We first recruit a team of at least 10 robots (including the initial awake robot) in the cone ending at the middle of the arc of the cone Figure 5(a) (2) we partition the disk into 8 semi-cones with a specific sequence of angles and 2 domes. Each robot wakes up in parallel each of the regions (see Figure 5(b)).

Lemma 3 *There exists a strategy to wake up asleep robots in a dome of angle α and radius one in a time at most $\alpha/2 + \sin(\alpha/2) + \phi(1 - \cos(\alpha/2))$ with an awake robot on the corner of the dome.*

Let a and b be the length of two chords of a semi-cone, where $a \leq b$ and α be the angle between them. To upper bound the makespan of asleep robots with one active robot on the apex, one can simply enclose a semi-cone with a larger cone and apply the Lemma 1 to obtain a makespan of $b + b\alpha\phi$ (see Figure 7 (a)). In

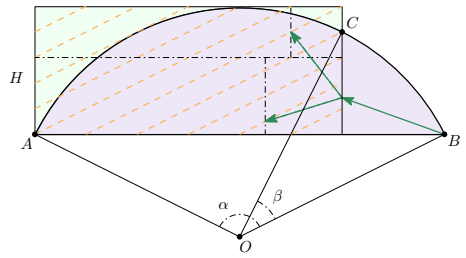
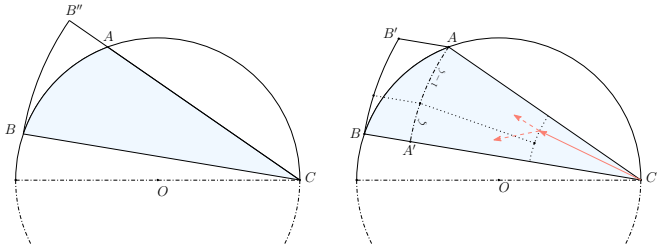


Figure 6: A dome of angle α. Illustration of proof of Lemma 3. Solving dome(α) using an enclosing rectangle.



(a) (b)

Figure 7: Illustration of proof of Lemma 4. (a) a semi-cone. (b) bounding a semi-cone with a simpler geometric object.

the following lemma, we state an upper bound $b + a\alpha\phi$ when $\alpha < \frac{\pi}{4}$.

Lemma 4 *There exists a strategy to wake up robots in a semi-cone of angle $\alpha < \frac{\pi}{4}$ and side lengths a and b where $a < b$, in time at most $b + a\alpha\phi$, where the initial active robot starts at the apex of the semi-cone.*

By having an upper bound for the makespan of a dome and semi-cone, we are ready to improve the lower bound on the number of robots needed to ensure a makespan of $1 + 2\sqrt{2}$.

Theorem 3 *Let $n \geq 281$ be an integer. There exists a strategy to wake up n robots in a unit disk in time less than $1 + 2\sqrt{2}$.*

7 A Better Upper-bound On The Wake-Up Time

In this section, we present an improved approximation on the makespan of $n \in \mathbb{N}$ asleep robots located in a unit disk on Euclidean plane. By a careful study of the first step in the analysis of Lemma 2 one can propose the following improvement on the monotonic strategy mentioned for the unit crown of width w and angle θ.

Lemma 5 *There exists a strategy to wake up all of the robots in a crown of angle θ and width w starting with two awake robots at a corner on the exterior side of the crown in time at most $\theta + \left(\frac{\phi^4}{\phi^3+\theta}\right) w$.*

Similar to Corollary 1, one can obtain the following improved corollary for the makespan of robots starting with one active robot at a corner of the interior side of the crown.

Let q be the first asleep robot in an angular sweep starting from the active robot p_0. If p_0 first awakes q and then the two robots directly join at the exterior side of the crown, we can then apply Lemma 1 to the rest of the crown and obtain the following corollary:

Corollary 3 *There exists a strategy to wake up all of the robots in a crown of angle $\theta \leq \pi$ and width w starting with one awake robot at a corner on the interior side of the crown in time at most $crown(w, \theta) \leq \theta + \left(1 + \frac{\phi^4}{\phi^3+\theta}\right) w$.*

In the rest of this section, assume that the asleep robots $p_1, p_2, \cdots, p_n$, are in sorted order based on their distance from p_0. That is, if d_i is the distance of p_i from p_0, for each $1 \leq i \leq n$, then we have $d_1 \leq d_2 \leq d_3 \leq \cdots \leq d_n$. We introduce two basic strategies to wake up the robots in set S, and by mixing these two strategies, we get an upper bound for the makespan of n robots located in a unit disk in the plane.

Strategy One

The first awake robot at the origin, p_0, travels to the closest robot, p_1, at a distance of d_1, and after activating p_1, both of the robots, p_0 and p_1, travel back to the origin at a total time of $2d_1$. Recall that the point at a distance of d_2 from the origin is p_2. Let ℓ be the line that passes through the origin and p_2. Next, p_0 and p_1 follow different paths. p_0 travels a distance of d_2 to activate p_2. Then, p_0 and p_2 split the remaining region into two equal crowns of angle $\frac{2\pi}{3}$ and width $1 - d_2$ (See Figure 8). Simultaneously, p_1 uses a strategy as in the proof of Corollary 3 to wake up all the robots within the crown of angle $\frac{2\pi}{3}$ and width $1 - d_2$ with the bisector ℓ. Using Corollary 3, each of p_0, p_1 and p_2 wake up their designated crown in time of at most $\frac{2\pi}{3} + \left(1 + \frac{\phi^4}{\phi^3+\frac{2\pi}{3}}\right)(1 - d_2)$.

Note that $\frac{2\pi}{3} + \left(1 + \frac{\phi^4}{\phi^3+\frac{2\pi}{3}}\right)(1 - d_2)$ is decreasing as function of d_2. Since $d_1 \leq d_2$, the total makespan of this strategy, $T_1(d_1)$, as a function of d_1, is upper bounded by:

$$T_1(d_1) \leq 2d_1 + d_2 + crown(1 - d_2, 2\pi/3)$$

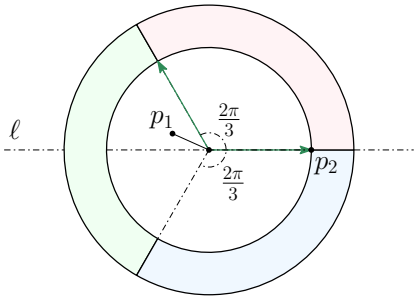


Figure 8: Strategy one. Three crowns of angle $\frac{2\pi}{3}$ and width $1 - d_1$.

$$\leq 1 + 2d_1 + \frac{2\pi}{3} + \left(\frac{\phi^4}{\phi^3 + \frac{2\pi}{3}}\right)(1 - d_2)$$

$$\leq 1 + 2d_1 + \frac{2\pi}{3} + \left(\frac{\phi^4}{\phi^3 + \frac{2\pi}{3}}\right)(1 - d_1).$$

Strategy Two

Note that strategy one is *good* when d_1 is small. We use a simpler idea for the case when d_1 is large, i.e., the robots are close to the boundary of the unit disk. In this strategy, the first awake robot, p_0, travels to the closest point p_1 at a distance of d_1. Note that after activating p_1, the disk of radius d_1 centered at the origin has no robots to be activated. Next, p_0 and p_1 split the remaining region into two crowns of angle π and width $1 - d_1$. Therefore, using Corollary 3, the total makespan of the second strategy, $T_2(d_1)$, as a function of d_1, is upper bounded by:

$$T_2(d_1) \leq d_1 + crown(1 - d_1, \pi) \leq 1 + \pi + \left(\frac{\phi^4}{\phi^3 + \pi}\right)(1 - d_1).$$

Best of Two Worlds

By analyzing the best makespan of these 2 strategies, we obtain the following result:

Theorem 4 *Let n be a non-negative integer. There exists a strategy to wake up n robots within a unit disk starting with an awake robot in the center in time less than 4.6211. The construction of such a wake-up tree can be done in linear time.*

It is worth mentioning that a wake-up tree of depth at most 4.6211 can be done in linear time. Whenever n is large, the construction of such a wake-up tree can be done in linear time using Linear-Split-Strategy using a partition of the disk into cones and applying

linear time for every cone (cf. Appendix B of [4]). These technical constructions are based on binary heaps and do not require any ordering.

Acknowledgement

This research was conducted at LaBRI, University of Bordeaux, during the fall of 2023, while the fourth author was visiting the other authors. The fourth author sincerely appreciates the warm hospitality and the academically enriching atmosphere at LaBRI. Furthermore, gratitude is extended to the University of Ottawa for supporting this visit through the PhD mobility program in France.

References

[1] Z. Abel, H. A. Akitaya, and J. Yu. Freeze tag awakening in 2D is NP-hard. *In Abstracts from the 27th Fall Workshop on Computational Geometry*, pages 105–107, 2017.

[2] E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. Mitchell, and M. Skutella. The freeze-tag problem: how to wake up a swarm of robots. *Algorithmica*, 46:193–221, 2006.

[3] E. M. Arkin, M. A. Bender, and D. Ge. Improved approximation algorithms for the freeze-tag problem. In A. L. Rosenberg and F. M. auf der Heide, editors, *SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003)*, pages 295–303. ACM, 2003.

[4] N. Bonichon, A. Casteigts, C. Gavoille, and N. Hanusse. Freeze-tag in L_1 has wake-up time five, arXiv:2402.03258, 2024.

[5] J. Brunner and J. Wellman. An optimal algorithm for online freeze-tag. In M. Farach-Colton, G. Prencipe, and R. Uehara, editors, *10th International Conference on Fun with Algorithms, FUN 2021, May 30 to June 1, 2021, Favignana Island, Sicily, Italy*, volume 157 of *LIPIcs*, pages 8:1–8:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[6] L. de Oliveira Silva. Freeze-tag is NP-hard in 3D with L_1 distance. *CoRR*, abs/2301.07757, 2023.

[7] E. D. Demaine and M. Rudoy. Freeze tag is hard in 3D. *In Abstracts from the 27th Fall Workshop on Computational Geometry*, page 108–110, 2017.

[8] M. Hammar, B. J. Nilsson, and M. Persson. The online freeze-tag problem. In J. R. Correa, A. Hevia, and M. A. Kiwi, editors, *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*, volume 3887 of *Lecture Notes in Computer Science*, pages 569–579. Springer, 2006.

[9] J. Könemann, A. Levin, and A. Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica*, 41(2):117–129, 2005.

[10] L. L. C. Pedrosa and L. de Oliveira Silva. Freeze-tag is np-hard in 3d with l_1 distance. In C. G. Fernandes and S. Rajsbaum, editors, *Proceedings of the XII Latin-American Algorithms, Graphs and Optimization Symposium, LAGOS 2023, Huatulco, Mexico, September 18-22, 2023*, volume 223 of *Procedia Computer Science*, pages 360–366. Elsevier, 2023.

[11] M. O. Sztainberg, E. M. Arkin, M. A. Bender, and J. S. B. Mitchell. Theoretical and experimental analysis of heuristics for the "freeze-tag" robot awakening problem. *IEEE Trans. Robotics*, 20(4):691–701, 2004.

[12] E. N. Yazdi, A. Bagheri, Z. Moezkarimi, and H. Keshavarz. An o(1)-approximation algorithm for the 2-dimensional geometric freeze-tag problem. *Inf. Process. Lett.*, 115(6-8):618–622, 2015.

Hyperplane Distance Depth*

Stephane Durocher† Amirhossein Mashghdoust†

Abstract

Depth measures quantify central tendency in the analysis of statistical and geometric data. Selecting a depth measure that is simple and efficiently computable is often important, e.g., when calculating depth for multiple query points or when applied to large sets of data. In this work, we introduce *Hypaerplane Distance Depth (HDD)*, which measures the centrality of a query point q relative to a given set P of n points in $\mathbb{R}^d$, defined as the sum of the distances from q to all $\binom{n}{d}$ hyperplanes determined by points in P. We present algorithms for calculating the HDD of an arbitrary query point q relative to P in $O(d \log n)$ time after preprocessing P, and for finding a median point of P in $O(dn^d \log n)$ time. We study various properties of hyperplane distance depth, and show that it is convex, symmetric, and vanishing at infinity.

1 Introduction

Depth measures describe central tendency in statistical and geometric data. A median of a set of univariate data is a point that partitions the set into two halves of equal cardinality, with smaller values in one part, and larger values in the other. Various definitions of medians exist in higher dimensions (multivariate data), seeking to generalize the one-dimensional notion of median (e.g., [6]). For geometric data and sets of geometric objects, applications of median-finding include calculating a centroid, determining a balance point in physical objects, and defining cluster centers in facility location problems [7]. A median is frequently used in statistics to describe the central tendency of a data set. It is particularly useful when dealing with skewed distributions or datasets that contain outliers. By using a median, analysts can obtain a representative value that is less affected by extreme values and outliers [10].

In 1975, Tukey introduced the concept of data depth for evaluating centrality in bivariate data sets [12]. The depth of a particular query point q in relation to a given set P gauges the extent to which q is situated within the overall distribution of P; i.e., when q's depth is large, q tends to be near the center of P. Since the introduction of Tukey depth (also called half-space depth), many more depth functions have been proposed.

Data depth functions should ideally satisfy specific properties, such as *convexity*, *stability* (small perturbations in the data do not result in large changes in depth values), *robustness* (depth is not heavily influenced by outliers or extreme values in the data), *affine invariance* (the depth function remains consistent under linear transformations of the data, such as translation, scaling, and rotation), *maximality at the center* (points closer to the geometric center of the data set have higher depth values), and *vanishing at infinity* (depth values approach zero as a query point moves away from the data set) [14].

2 Related Work

Tukey [12] first introduced the concept of location depth. In $\mathbb{R}^2$, the Tukey depth of a point $q \in R^2$ relative to a set P of n points in $\mathbb{R}^2$ is defined as the smallest number of points of P on one side of a line passing through q. This concept can also be generalized to higher dimensions.

Definition 1 (Tukey Depth [12]) *The* Tukey depth *of a point $q \in \mathbb{R}^d$ relative to a set P of points in $\mathbb{R}^d$, is the minimum number of points of P in any closed half-space that contains q.*

In univariate space, e.g., in $\mathbb{R}$, the Tukey depth of q is determined by considering the minimum of the count of points $p_i \in P$ where $p_i < q$, and the count of points $p_i \in P$ where $p_i > q$.
A *Tukey median* of a set P in $\mathbb{R}^d$ corresponds to a point (or points) with maximum Tukey depth among all points in $\mathbb{R}^d$.

Since Tukey's introduction of Tukey depth, several other important depth functions have been defined to measure the centrality of q relative to P.

Definition 2 (Mahalanobis Depth [9]) *The* Mahalanobis depth *of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is defined as $[1 + (q - \bar{q})^T {P_d}^{-1} (q - \bar{q})]^{-1}$, where $\bar{q}$ and P_d are the mean vector and dispersion matrix of P.*

This function lacks robustness, as it relies on non-robust measures like the mean and the dispersion matrix. Another possible disadvantage of Mahalanobis depth is its reliance on the existence of second moments [9].

*This work is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†Department of Computer Science, University of Manitoba, {stephane.durocher,amirhossein.mashghdoust}@umanitoba.ca

Definition 3 (Convex Hull Peeling Depth [2]) *The* convex hull peeling depth *of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is the level of the convex layer to which q belongs.*

A convex layer is established by recursively removing points on the convex hull boundary of P until q is outside the hull. Begin by constructing the convex hull of P. Points of P on the boundary of the hull constitute the initial convex layer and are removed. Then, form the convex hull anew with the remaining points of P. The points along this new hull's boundary constitute the second convex layer. This iterative process continues, generating a sequence of nested convex layers. The deeper a query point q resides within P, the deeper the layer it belongs to. However, the method of convex hull peeling depth possesses certain drawbacks. It fails to exhibit robustness in the presence of outliers or noise. Additionally, it's unfeasible to associate this measure with a theoretical distribution.

Definition 4 (Oja Depth [11]) *The* Oja depth *of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is defined as the sum of the volumes of every closed simplex having one vertex at q and its remaining vertices at any points of P.*

In $\mathbb{R}^2$, the Oja depth of a point q is the sum of the areas of all triangles formed by the vertices q,p_i, and p_j, where $\{p_i, p_j\} \subseteq P$.

Definition 5 (Simplicial Depth [8]) *The* simplicial Depth *of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is defined as the number of closed simplices containing q and having $d+1$ vertices in P.*

The simplicial depth of a point $q \in \mathbb{R}^2$ is the number of triangles with vertices in P and containing q. This is a common measure of data depth.

Definition 6 (L_1 Depth [13]) *The* L_1 depth *of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is defined as $\sum_{p_i \in P} ||p_i - q||_1$.*

The L_1 Median is the point that minimizes the sum of the absolute distances (also known as the L_1 norm or Manhattan distance) to all other points in P. The key advantage of the L_1 Median is its robustness to outliers. It is less sensitive to extreme values in the dataset compared to the L_2 Median, which minimizes the sum of squared distances. As a result, the L_1 Median can provide a more accurate estimate of central tendency in datasets with outliers or heavy-tailed distributions. The L_1 Median is used in various fields, including finance, image processing, and robust statistics, whenever there is a need for a robust estimate of the central location of a dataset that may contain atypical values.

Definition 7 (L_2 Depth [14]) *The* L_2 depth *(mean) of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is defined as $\sum_{p_i \in P} ||p_i - q||^2$.*

The L_2 Median is the point that minimizes the sum of the squared Euclidean distances. The mean is a widely used measure of central tendency in statistics and data analysis. The mean is not robust to outliers; a single outlier can pull the mean arbitrarily far.

Definition 8 (Fermat-Weber Depth [4]) *The* Fermat-Weber depth *(Geometric depth) of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is defined as $\sum_{p_i \in P} ||p_i - q||$.*

A deepest point (median) with respect to Fermat-Weber depth cannot be calculated exactly in general when $d \geq 2$ and $|P| \geq 5$ [1].

There is no single depth function that universally outperforms all others. The choice of a particular depth function often depends on its suitability for a specific dataset or its ease of computation. Nevertheless, there are several desirable properties that all data depth functions should ideally possess. In Section 3, we introduce a new depth measure, and we examine which of these properties it satisfies.

3 Results

In this section, we will introduce the Hyperplane Distance Depth (HDD) measure and study its properties.

3.1 Defintion

Definition 9 (Hyperplane distance depth) *The Hyperplane distance depth (HDD) of a point $q \in \mathbb{R}^d$ relative to a set P in $\mathbb{R}^d$ is defined as*

$$D_P(q) = \sum_{h_i \in H_P} \text{dist}(q, h_i), \tag{1}$$

where H_P is the set of all $\binom{n}{d}$ $(d-1)$-dimensional hyperplanes determined by points in P, and $\text{dist}(q, h_i)$ denotes the Euclidean (L_2) distance from the point q to the hyperplane h_i.

Both Fermat-Weber depth and hyperplane distance depth are defined as sums of Euclidean (L_2) distances. Unlike Fermat-Weber depth, for which the location of a median cannot be computed exactly in general when $d \geq 2$ [1], as we show in Section 4, the location of a HDD median can be computed exactly.

3.2 Properties

Theorem 1 *In $\mathbb{R}$, the HDD median relative to the set P coincides with the usual univariate definition of median.*

Proof. By Definition 9, the median is a point that minimizes the sum of the distances to all possible points passing through each point in P. Therefore, $H_P = P$. Consequently, the HDD median is equivalent to the usual definition of median in a one-dimensional space. □

Theorem 2 *The HDD function $D_P(q)$ relative to the set P is convex over $q \in \mathbb{R}^d$.*

Proof. The distance function $d_{h_i}(q)$ from a query point q to the hyperplane h_i is convex. Any non-negative linear combination of convex functions is convex. Therefore, the HDD function $\sum_{h_i \in H_P} d_{h_i}(q) = D_P(q)$ is convex over q. □

Theorem 3 *The HDD median point relative to the set P of points in $\mathbb{R}^d$ is always on one of the intersection points between d hyperplanes in H_P.*

Proof. The distance from the point $q \in \mathbb{R}^d$ to a hyperplane h_i is equal to $d_{h_i}(q) = \frac{|w_i.q+b_i|}{\|w_i\|}$ where w_i and b_i are the hyperplane's normal vector and the offset respectively. Therefore, the HDD of the point q is equal to

$$D_P(q) = \sum_{h_i \in H_P} d_{h_i}(q) = \sum_{h_i \in H_P} \frac{|w_i.q + b_i|}{\|w_i\|} \tag{2}$$

Depending on the position of q with respect to h_i, $d_{h_i}(q) = \frac{|w_i.q+b_i|}{\|w_i\|}$ can be equal to $+\frac{w_i.q+b_i}{\|w_i\|}$ (above the hyperplane) ,$-\frac{w_i.q+b_i}{\|w_i\|}$ (below the hyperplane), or 0 (on the hyperplane). Therefore, for any point q we have

$$\sum_{h_i \in H_P} d_{h_i}(q) = \sum_{h_i \in H_P} g_{i,q} \frac{w_i.q + b_i}{\|w_i\|}$$
$$g_{i,q} = \begin{cases} +1, & \text{if } q \text{ is above } h_i \\ -1, & \text{if } q \text{ is below } h_i \\ 0, & \text{if } q \text{ is on } h_i \end{cases} \tag{3}$$

It is worth noting that the derivative of the equation (3) exists if q is not on any of the hyperplanes in H_P ($g_{i,q} \neq 0$). Now to find the HDD median with the minimum HDD measure, we should compute the derivative with respect to q and see where it will be equal to 0. For any query point q inside a region bounded by some H_P hyperplanes and not on any H_P hyperplanes (Figure 1) we have

$$\frac{d}{dq} D_P(q) = \frac{d}{dq} \sum_{h_i \in H_P} g_{i,q} \frac{w_i.q + b_i}{\|w_i\|} = \sum_{h_i \in H_P} g_{i,q} \frac{w_i}{\|w_i\|} \tag{4}$$

Equation (4) above cannot be equal to 0 in general since there are no variables (4). This means the assumption

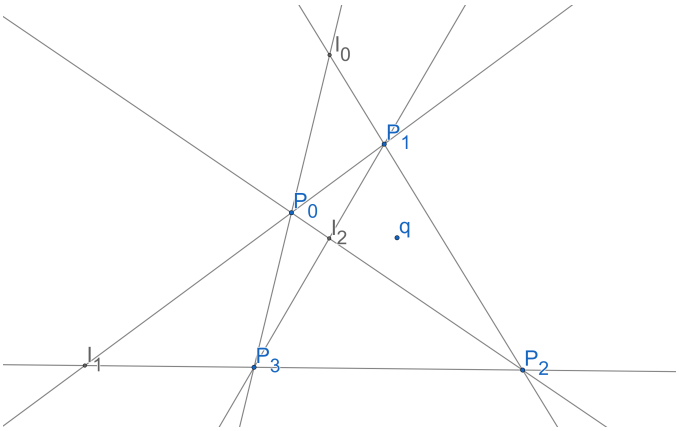


Figure 1: Example of HDD in two dimensions: $P = \{P_0, P_1, P_2, P_3\}$ is the set of input points, $I = \{I_0, I_1, I_2\}$ is the set of intersection points, and q is the query point.

we made about the query point not being on the hyperplanes in H_P was incorrect. Therefore, we can say the median is surely on one of the hyperplanes. If q is on h_j, $w_j.q + b_j$ will be equal to 0. Therefore, we can say

$$\frac{d}{dq} D_P(q) = \frac{d}{dq} \sum_{h_i \in H_P - \{h_j\}} g_{i,q} \frac{w_i.q + b_i}{\|w_i\|} = \sum_{h_i \in H_P - \{h_j\}} g_{i,q} \frac{w_i}{\|w_i\|}. \tag{5}$$

Using the same logic we can conclude that the median point should be on another hyperplane in addition to h_j. We can repeat these steps d times and after that, it will be proved that the median should be on the intersection point of d hyperplane (that will be a single point), thus the median will be on one of the intersection points. □

Theorem 4 *The HDD median point relative to the set P in $\mathbb{R}^d$ is always in the convex hull of the input points P.*

Proof. Let $D'_{p_i}(q)$ be the sum of the distances to all the hyperplanes in H_P passing through the point p_i. The minimum of this convex function is always on the point p_i where the HDD is equal to 0. On the other hand, since each hyperplane includes d input points from P, we have $D_P(q) = d \sum_{p_i \in P} D'_{p_i}(q)$.
Now consider a point q_o outside of the convex hull. by computing the gradient of $D_P(q_o)$, we will show that by moving q_o closer to the convex hull, the HDD gets smaller. Using the equation above we have $-\nabla H_P(q_o) = -d \sum_{p_i \in P} \nabla D'_{p_i}(q_o)$. Since the minimum of the function $D'_{p_i}(q)$ is on p_i, $-\nabla D'_{p_i}(q)$ is a vector pointing to p_i for $p_i \in P$. Therefore we can conclude that for every point q_o outside of the convex hull, $-\nabla H_P(q_o)$ points to the convex hull that means by moving toward that direction, the HDD decreases.

Therefore the HDD median is always in the convex hull of P. □

Theorem 5 *The HDD median point relative to the set P in $\mathbb{R}^d$ is always at the center of symmetry.*

Proof. Let p_M be the median of the P s.t. P is symmetric. If p_M is not on the center of symmetry, consider p'_M, the reflection of p_M across the center of symmetry. Because of the symmetry, it is trivial that the HDD measure of both points is equal. Since the median point has the minimum depth measure among the other points and the depth measure function is convex, all the points on the line segment $p_m p'_M$ should have a depth measure equal to the median. Therefore, the median is always at the center of symmetry. □

Theorem 6 *The HDD measure relative to the set P in $\mathbb{R}^d$ vanishes as we move the query point to infinity.*

Proof. As we move the query point q to infinity, it is straightforward that there exists a hyperplane $h_i \in H_P$ that gets further from q. Since we can move q arbitrarily far from h_i, and the distance from q to the remaining hyperplanes in H_P is non-negative, therefore HDD vanishes at infinity. □

Note that some measures of depth are defined such that deep points have high depth values and outliers have low depth values, whereas this property is reversed for other depth measures. HDD is of the latter type, with central points having a low sum of distances to hyperplanes in H_P, whereas this sum approaches infinity as q moves away from P. Consequently, for HDD, "vanishing at infinity" means that depth approaches ∞ as opposed to 0.

Theorem 7 *The HDD measure relative to the set P in $\mathbb{R}^d$ is not robust.*

Proof. We will prove this fact using a counter-example in a 2-dimensional space (Figure 2). We can move the HDD median by changing the location of 2 points which means the HDD is not robust. The median is always on one of the intersection points and we can place the points in a way that I_0 is always the median (Figure 2). We will compute the depth measures for the points I_0 (5) and I_i (7), where I_i is an arbitrary intersection point except I_0.

$$D_P(I_0) = \sum_{i\in[3,n]} d_{l_{P_1P_i}}(I_0) + \sum_{i\in[3,n]} d_{l_{P_2P_i}}(I_0) \tag{6}$$

$$D_P(I_i) = \sum_{i\in[3,n]} d_{l_{P_1P_i}}(I_i) + \sum_{i\in[3,n]} d_{l_{P_2P_i}}(I_i) + \binom{n-2}{2} d_{l_{P_3P_n}}(I_i) + d_{l_{P_1P_2}}(I_i) \tag{7}$$

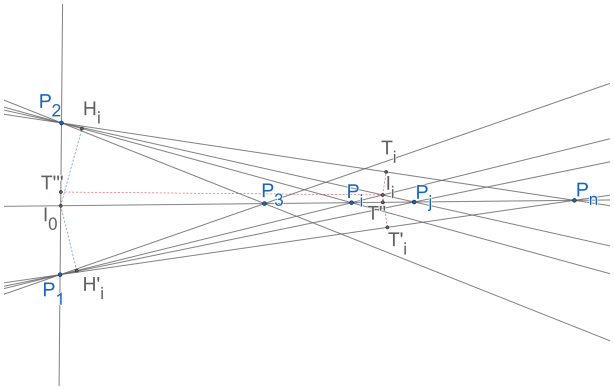


Figure 2: A counter-example that shows the HDD is not robust

Regardless of the I_0 position, we know that $I_0H_i < I_0P_2$ and $I_0H'_i < I_0P_1$. Therefore, we have (Equation (6)):

$$D_P(I_0) < (n-2)I_0P_2 + (n-2)I_0P_1 = (n-2)P_1P_2 \tag{8}$$

On the other hand, using Equation (7) we have:

$$D_P(I_i) > d_{l_{P_1P_2}}(I_i) \tag{9}$$

Now by moving the points P_1 and P_2 far enough, let $d_{l_{P_1P_2}}(I_i) = (n-2)P_1P_2 + m$, where m is a positive number. Therefore, we have (inequality 9):

$$D_P(I_i) > (n-2)P_1P_2 + m \tag{10}$$

Combining the inequality 8 and 10 we have $D_P(I_i) > D_P(I_0)$.

Consequently, I_0 is the median. By increasing m, the median I_0 gets as far as we want. This means by moving P_1 and P_2, we can move the median point as much as we want. □

Definition 10 (k-stability [5]) *A depth measure D is k-stable if for all points q in $\mathbb{R}^d$, all sets P in $\mathbb{R}^d$, all $\epsilon > 0$, and all functions $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^d$ such that $\forall p$, $\text{dist}(p, f(p)) \leq \epsilon$,*

$$k \cdot |D(q,P) - D(f_\epsilon(q), f_\epsilon(P)| \leq \epsilon, \tag{11}$$

where $f_\epsilon(P) = \{f_\epsilon(p) \mid p \in P\}$.

That is, for any ϵ-perturbation of P and q, the depth of q relative to P changes by at most $k\epsilon$.

Theorem 8 *The HDD measure relative to the set P in $\mathbb{R}^d$ is not k-stable for any constant k.*

Proof. Choose any $k > 0$ and let $n = \max\{1, \lceil 1/k \rceil + 1\}$. Let P be a set of n points in $\mathbb{R}$ and let $q \in \mathbb{R}$ lie to the left of P. By moving all points of P one unit to the right ($\epsilon = 1$), the hyperplane depth of q relative to P increases by a factor of n, regardless of k. Thus HDD is not k-stable. □

Theorem 9 *The HDD function $D_P(q)$ relative to the set P in $\mathbb{R}^d$ is not equivariant under affine transformations.*

Proof. We will prove this theorem using a counter-example. Consider the set of points $P = \{p_0(0,0), p_1(4,0), p_2(2,1)\}$. Using Theorem 3 we can show that the median is on point $p_2(2,1)$. Now consider the set $P' = \{p'_0(0,0), p'_1(4,0), p'_2(2,5)\}$ that is P under the non-uniform affine transformation matrix $\begin{bmatrix}1 & 0\\ 0 & 5\end{bmatrix}$. Using theorem 3 and 5 It can be shown that the median is on the line $p'_0p'_1$ now. This means that the HDD median is not equivariant under affine transformation. □

As we now show, HDD is equivariant under similarity transformations, including translation, rotation, reflection and uniform scaling, since these preserve the shape of P.

Theorem 10 *The HDD function $D_P(q)$ relative to the set P in $\mathbb{R}^d$ is equivariant under the similarity transformations.*

Proof. For any rotation, reflection, or translation transformation f, the distance from the query point q to any hyperplane h_i remains unchanged. That is, for any point q and any hyperplane h_i, $\text{dist}(q, h_i) = \text{dist}(f(q), f(h_i))$.

For any uniform scaling transformation f with a scaling factor of k, distances between each pair of points will be multiplied by k after the transformation. Therefore it is easy to show that, for any query point q, the HDD will be multiplied by k after uniform transformation. Therefore, the median is equivariant under the uniform scaling transformation. □

4 Algorithms

In this section, we provide three algorithms: a) to compute HDD depth queries in $O(d \log n)$ time after $O(n^{2d^2+2d})$ preprocessing time, b) to find an HDD median point in $O(dn^{d^2} \log n)$ time, and c) to find an approximate HDD median. Let P be a set of n points in $\mathbb{R}^d$, and let H_P be the set of $\binom{n}{d}$ hyperplanes determined by d point in P.

4.1 HDD Query Algorithm

The hyperplane distance depth of a query point q relative to P can be computed by directly evaluating Equation (2) in $O\left(\binom{n}{d}\right) = O(n^d)$ time. We will present an algorithm that can calculate HDD in logarithmic time after preprocessing. First, to measure the HDD of q, we need to store some coefficients belonging to each polytope formed by hyperplanes in H_P.

Consider Equation (3). Let S_P be the set of all minimal polytopes determined by the arrangement of hyperplanes in H_P. For a query point q_k in a polytope $s_k \in S_P$, the coefficients $g_{i,q}$ for $h_i \in H_P$ are the same. Therefore, for any points q_k in s_k, we can simplify the summation in (3) in $O(n^d)$ time and find the 2 coefficients a_k and b_k such that

$$D_P(q_k) = \sum_{h_i \in H_P} g_{i,q_k} \frac{w_i.q_k + b_i}{\|w_i\|} = a_k q_k + b_k \qquad (12)$$

Using Euler's characteristic theorem we know that there are $O(n^{d^2})$ polytopes formed by the hyperplanes in H e.g. in Figure 1 there are 18 polytopes (faces) formed by the 6 hyperplanes (lines). Therefore we will need $O(2n^{d^2}) \in O(n^{d^2})$ space and $O(n^{d^2}n^d) \in O(n^{d^2+d})$ time to preprocess.

Using the mentioned data structure we can calculate the HDD measure in $O(1)$ time if we know to which polytope the query point belongs.

Given n hyperplanes in d-dimensional space and a query point q, it takes $O(\log n)$ time to find the q location with a data structure of size $O(n^d)$ and a preprocessing time of $O(n^{2d+2})$[3]. In our problem, there are $\binom{n}{d} \in O(n^d)$ hyperplanes. Therefore, with a preprocessing time of $\binom{n}{d}^{2d+2} \in O(n^{2d^2+2d})$ and a space of $O(n^{d^2})$, we can find the location of q in $O\left(\log \binom{n}{d}\right) \in O(d \log n)$ time.

Now after finding the q's location in $O(d \log n)$, we can calculate the HDD measure $D_P(q_k)$ in $O(1)$ using Equation (12).

Therefore, after $O(n^{d^2+d} + n^{2d^2+2d}) \subseteq O(n^{2d^2+2d})$ preprocessing time using $O(n^{d^2})$ space, we can find the HDD of an arbitrary query point in $O(d \log n)$ time. This proves the following theorem.

Theorem 11 *We can preprocess any given set P of n points in $\mathbb{R}^d$ in $O(n^{2d^2+2d})$ time, such that given any point $q \in \mathbb{R}^d$, we can compute $D_P(q)$ in $O(d \log n)$ time.*

4.2 Finding a HDD Median

By Theorem 3, a straightforward algorithm for finding an HDD median of P is to check all points of intersection between d hyperplanes in H_P using an exhaustive search. There are $\binom{n}{d}$ hyperplanes in H_P and therefore $\binom{\binom{n}{d}}{d} \in O(n^{d^2})$ intersection points between hyperplanes in H_P. Since it takes $O(n^d)$ to compute the Equation (2) directly, a HDD median of P can be found in $O(n^{d^2+d})$ time by this brute-force algorithm.

Next, we will introduce an algorithm that finds the HHD median in $O(dn^{d^2} \log n)$ time. When $d = 2$, this second algorithm runs in $O(n^4 \log n)$ time, compared to $O(n^6)$ time for the brute-force algorithm. First, we will show that we can find the point with the smallest HDD

on a line in $O(dn^d \log n)$ time. Consider the intersection of $d-1$ hyperplanes in H_P that determine a line ℓ. Since every hyperplane in H_P has exactly one point of intersection with ℓ, $H_P \cap \ell$ is a set of $O(n^d)$ points of intersection. By Theorem 2, we can conclude that the hyperplane depth of points on ℓ is a convex function. Since $H_P \cap \ell$ is discrete, using binary search and calculating HDD in $O(n^d)$ time using Equation (2), we can find the intersection point with the minimum HDD in $O(n^d \log(n^d)) = O(dn^d \log n)$ time.

We can use the algorithm above to find the minimum point for each intersection line among hyperplanes in H_P to find an HDD median. Since each $d-1$ hyperplanes in H_P form a line, there are $\binom{n}{d-1} \in O(n^{d^2-d})$ lines and thus we can find the median in $O(dn^{d^2} \log n)$ time. This proves the following theorem.

Theorem 12 *Given a set P of n points in $\mathbb{R}^d$, we can find an HDD median of P in $O(dn^{d^2} \log n)$ time.*

4.3 Finding an Approximate HDD Median in $\mathbb{R}^2$

In this section, we will present an approximation algorithm to find an HDD median of P with an error of $\frac{a\sqrt{2}}{2^{\frac{m}{2}+1}}$ in $O(mn^2 \log n)$ time, for any fixed $m \in \mathbb{Z}^+$, where a is the diameter of P.

Theorem 13 *Given a set P of n points in $\mathbb{R}^2$, in $O(mn^2 \log n)$ time we can find a point x' in $\mathbb{R}^2$ such that $\text{dist}(x', x) \leq \frac{a\sqrt{2}}{2^{\frac{m}{2}+1}}$, for any fixed $m \in \mathbb{Z}^+$, where x denotes an HDD median of P and $a = \max_{p,q \in P} \text{dist}(p, q)$.*

Proof. Let l_a be an arbitrary line among the lines in H_P (see Figure 3). There are $\binom{n}{2}$ lines in H_P and, consequently, $O(n^2)$ points of intersection between l_a and lines in H_P. Using an analogous argument as in the proof of Theorem 3, the point with minimum HDD on l_a lies at an intersection of l_a and a line in H_P. Therefore, using the same algorithm described in Section 4.2, we can find the point $i_{\min}$ on l_a with minimum HDD in $O(n^2 \log n)$ time; let $h_{\min}$ denote the line in H_P such that $i_{\min} = h_{\min} \cap l_a$. Next we find the closest points of intersection in H_P to $i_{\min}$ on the line $h_{\min}$ in each direction, say I_u and I_d. We compute the HDD for all the three points $i_{\min}$, I_d, and I_u. Since $i_{\min}$ has minimum HDD on the line l_a, if $D_P(i_{\min}) < \min\{D_P(I_u), D_P(I_d)\}$, then $i_{\min}$ is the HDD median (by Theorem 2). By Theorem 2 again, $D_P(i_{\min}) < D_P(I_u)$ or $D_P(i_{\min}) < D_P(I_d)$. Furthermore, $D_P(i_{\min}) > D_P(I_u)$ or $D_P(i_{\min}) > D_P(I_d)$. Without loss of generality, suppose $D_P(I_d) < D_P(i_{\min}) < D_P(I_u)$. We claim that all points in the half-plane bounded by $h_{\min}$ that contains I_u have HDD that exceeds $D_P(i_{\min})$; we prove this by contradiction. Suppose there exists a point A in this half-plane such that $D_P(A) < D_P(i_{\min})$. Let B be the intersection point of the line l_a and the line segment $\overline{AI_d}$. Since $i_{\min}$ has minimum HDD on the line l_a, therefore, $D_P(i_{\min}) < D_P(B)$. Furthermore, $D_P(A) < D_P(B)$. On the other hand, we assumed $D_P(I_d) < D_P(i_{\min}) < D_P(I_u)$ and we know $D_P(i_{\min}) < D_P(B)$. Consequently, $D_P(I_d) < D_P(B)$. Combining the two resulting inequalities above, we have $D_P(A) < D_P(B)$ and $D_P(I_d) < D_P(B)$, which is impossible since the three points are on the same line and the HDD function is convex. Therefore, no such point A can exist.

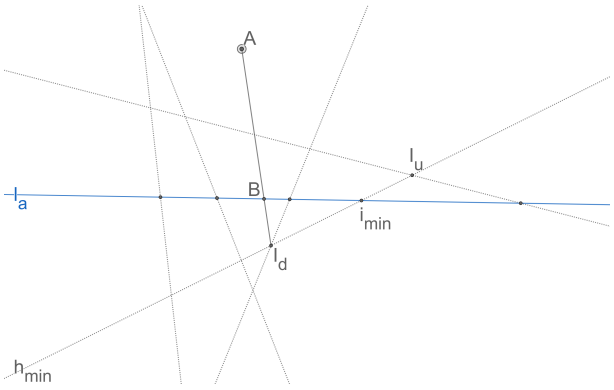


Figure 3: An algorithm to eliminate the points belonging to a half-space. The blue line l_a is an arbitrary line dividing the space into 2 halves. The dotted gray lines are the lines in H_P.

Therefore, no point of intersection in H_P in this half-plane can be an HDD median of P; in $O(n^2 \log n + 3n^2) \subseteq O(n^2 \log n)$ time we can remove these points from consideration in our search for a median.

Now we will use this property to approximate the median point. Firstly, we will find the diameter a of the input points in $O(n)$ time and consider an $a \times a$ square that contains P (see Figure 4). By Theorem 4, we know that the median lies inside this square. At each step, we draw the two lines ON and OM that partition the square into four similar smaller squares, each with dimensions $\frac{a}{2} \times \frac{a}{2}$, and we apply the above algorithm to eliminate two half-planes in $O(n^d \log n)$ time. After m steps we have a square of dimensions $\frac{a}{2^m} \times \frac{a}{2^m}$ and we return its center as an approximation of the HDD median. Since the HDD median is a point inside this square, the error is at most $\frac{a\sqrt{2}}{2^{m+1}}$. The total time complexity of the algorithm is $O(mn^2 \log n)$. □

This strategy can be generalized to higher dimensions by finding the minimum HDD on an arbitrary hyperplane h_a (analogous to the line l_a in the proof of Theorem 13) to eliminate a half-space, but the time complexity of finding the minimum HDD point on h_a is high.

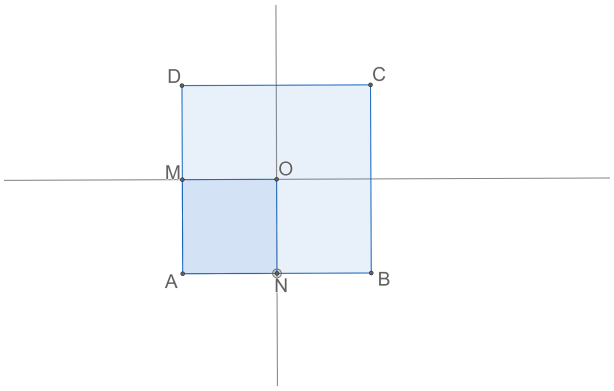


Figure 4: Illustration in support of Theorem 13

5 Discussion and Possible Directions for Future Research

Our algorithm for computing HDD queries presented in Section 4.1 requires $O(n^{d^2})$ space and $O(n^{2d^2+2d})$ preprocessing time. One natural possible direction for future research is to identify algorithms with improved preprocessing time or space.

Our algorithm for finding an HDD median presented in Section 4.2 requires $O(dn^{d^2} \log n)$ time. In addition to seeking to identify lower bounds on the worst-case running time required to find an HDD median, we could attempt to reduce the running time using techniques such as gradient descent or linear programming.

Our analysis of our algorithm for finding an approximate HDD median presented in Section 4.3 does not capitalize on the fact that the number of candidate points decreases on each step; we charge $O(n^2 \log n)$ time per step. If it could be shown that a constant fraction of the remaining points are eliminated on each step, then the bound on the algorithm's time complexity would be significantly improved.

Finally, we could consider alternative definitions for depth using similar notions to those in Definition 1. E.g., one can define a "line distance depth" that evaluates the distances to all possible lines passing through each pair of points in the set of input points. This definition coincides with Definition 1 when $d \leq 2$, but differs in higher dimensions, for $d \geq 3$.

References

[1] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete and Computational Geometry*, 3:177–191, 1988.

[2] V. Barnett. The ordering of multivariate data. *Journal of the Royal Statistical Society: Series A (General)*, 139(3):318–344, 1976.

[3] B. Chazelle and J. Friedman. Point location among hyperplanes and unidirectional ray-shooting. *Computational Geometry*, 4(2):53–62, 1994.

[4] R. Durier and C. Michelot. Geometrical properties of the Fermat-Weber problem. *European Journal of Operational Research*, 20(3):332–343, 1985.

[5] S. Durocher and D. Kirkpatrick. The projection median of a set of points. *Computational Geometry: Theory and Applications*, 42(5):364–375, 2009.

[6] S. Durocher, A. Leblanc, and M. Skala. The projection median as a weighted average. *Journal of Computational Geometry*, 8(1):78–104, 2017.

[7] R. Z. Farahani, M. SteadieSeifi, and N. Asgari. Multiple criteria facility location problems: A survey. *Applied mathematical modelling*, 34(7):1689–1709, 2010.

[8] R. Y. Liu. On a notion of data depth based on random simplices. *The Annals of Statistics*, pages 405–414, 1990.

[9] P. C. Mahalanobis. On the generalized distance in statistics. *Sankhyā: The Indian Journal of Statistics, Series A (2008-)*, 80:S1–S7, 2018.

[10] A. T. Murray and V. Estivill-Castro. Cluster discovery techniques for exploratory spatial data analysis. *International journal of geographical information science*, 12(5):431–443, 1998.

[11] H. Oja. Descriptive statistics for multivariate distributions. *Statistics & Probability Letters*, 1(6):327–332, 1983.

[12] J. Tukey. Mathematics and the picturing of data. In *Proc. Int. Cong. Math.*, pages 523–531, 1957.

[13] Y. Vardi and C.-H. Zhang. The multivariate l_1-median and associated data depth. *Proceedings of the National Academy of Sciences*, 97(4):1423–1426, 2000.

[14] Y. Zuo and R. Serfling. General notions of statistical depth function. *Annals of statistics*, 28(2):461–482, 2000.

Quantum Speedup for Some Geometric 3SUM-Hard Problems and Beyond

J. Mark Keil* Fraser McLeod† Debajyoti Mondal‡

Abstract

The geometric 3SUM-hard problems have widely been studied in computational geometry and recently, these problems have been examined under the quantum computing model. For example, Ambainis and Larka [TQC'20] designed a quantum algorithm that can solve many geometric 3SUM-hard problems in $O(n^{1+o(1)})$-time, whereas Buhrman [ITCS'22] investigated lower bounds under quantum 3SUM conjecture that claims there does not exist any sublinear $O(n^{1-\delta})$-time quantum algorithm, where $\delta > 0$, for the 3SUM problem. The main idea of Ambainis and Larka is to formulate a 3SUM-hard problem as a search problem, where one needs to find a point with a certain property over a set of regions determined by a line arrangement in the plane.

This paper further generalizes the technique of Ambainis and Larka for some 3SUM-hard problems when a solution may not necessarily correspond to a single point or the search regions do not immediately correspond to the subdivision determined by a line arrangement. Given a set of n points and a positive number q, we design $O(n^{1+o(1)})$-time quantum algorithms to determine whether there exists a triangle among these points with an area at most q or a unit disk that contains at least q points. We also give an $O(n^{1+o(1)})$-time quantum algorithm to determine whether a given set of intervals can be translated so that it becomes contained in another set of given intervals and discuss further generalizations.

1 Introduction

A rich body of research investigates ways to speed up algorithmic computations by using quantum computing techniques. Grover's algorithm [20] (a quantum search algorithm) has often been leveraged to obtain quadratic speedup for various problems over the classical solution. For example, consider the problem of finding a specific item within an unordered database of n items. In the classical setting, this task requires $\Omega(n)$ operations. However, with high probability, Grover's algorithm can find the item in $O(\sqrt{n})$ quantum operations [20]. In this paper we investigate quantum speedup for some geometric 3SUM-hard problems.

Given a set S of n numbers, the 3SUM problem asks whether there are elements $a, b, c \in S$ such that $a + b + c = 0$. The class of 3SUM-hard problems consists of problems that are at least as hard as the 3SUM problem. The classical 3SUM conjecture states that the class of 3SUM-hard problems does not admit a truly sub-quadratic $O(n^{2-\delta})$ time, where $\delta > 0$, in a classical computer [19]. Some logarithmic-factor speedups are now known [4, 11]. However, 3SUM can be solved in $O(n \log n)$ time in a quantum computer by applying Grover search over all possible pairs as follows [2]: We have $O(n)$ quantum search operations to resolve and if we maintain the elements of S in a balanced binary search tree, then for each pair a, b, we can decide the existence of $-(a + b) \in S$ in $O(\log n)$ time. In general, such straightforward quantum speedup does not readily apply to all problems even if they can be solved in $O(n^2)$ time in a classic computer [7, Table 1].

Quantum algorithms have previously been examined for many computational geometry problems [1, 23, 25, 26, 27, 28], but here we mainly focus on the class of 3SUM-hard problems. Ambainis and Larka [2] designed a quantum algorithm that can solve many geometric 3SUM-hard problems in $O(n^{1+o(1)})$-time. Some examples are Point-On-3-Lines, Triangles-Cover-Triangle, and Point-Covering. The Point-On-3-Lines problem takes a set of lines as input and asks to determine whether there is a point that lies on at least 3 lines. The Triangles-Cover-Triangle problem asks whether a given set of triangles in the plane covers another given triangle. Given a set of n half-planes and an integer t, the Point-Covering problem asks whether there is a point that hits at least t half-planes.

The idea of Ambainis and Larka [2] is to model these problems as a point search problem over a subdivision of the plane with a small number of regions. Specifically, consider a random set of k lines in the Point-On-3-Lines problem and a triangulation of an arrangement of these lines, which subdivides the plane into $O(k^2)$ regions. We can check each corner of these regions to check whether it hits at least three lines in $O(nk^2)$ time. Otherwise, we can search each region recursively by taking only the lines that intersect the region into consideration. It is known that with high probability every subproblem size (i.e., the number of lines intersecting a region) would be small [13, 21], and one can obtain a running

*Department of Computer Science, University of Saskatchewan, Saskatoon, Canada, mark.keil@usask.ca

†Department of Computer Science, University of Saskatchewan, Saskatoon, Canada, fdm360@mail.usask.ca

‡Department of Computer Science, University of Saskatchewan, Saskatoon, Canada, d.mondal@usask.ca.

time of $O(n^{1+o(1)})$ by a careful choice of k and by the application of Grover search [2]. For the Point-Covering problem, one can construct a similar subdivision of the plane using k random half-planes. We can then count for each region R, the number i of half-planes fully covering R in $O(nk^2)$ time, and if a solution is not found, then recursively search in the subproblem for a point that hits at least $(t-i)$ half-planes. For the Triangles-Cover-Triangle problem, one can construct the $O(k^2)$-size subdivision (of the given triangle T which we want to cover) by k lines determined by k segments that are randomly chosen from the boundaries of the set S of given triangles. In $O(nk^2)$ time we can determine the regions of T that are fully covered by a single triangle of S. For every remaining region R, let $S(R)$ be a set of triangles where each intersects R but does not fully contain R. We now can search over all such regions R recursively for a point that is not covered by $S(R)$.

In this paper we show how Ambainis and Larka's [2] idea can be adapted even for problems where a solution may not correspond to a single point or the search regions do not necessarily correspond to a subdivision determined by an arrangement of straight lines. Specifically, we show that the following problems admit an $O(n^{1+o(1)})$-time quantum algorithm.

q-Area Triangle: Given a set S of n points, decide whether they determine a triangle with area at most q.

q-Points in a Disk: Given a set S of n points, determine whether there is a unit disk that covers at least q of these points.

Interval Containment: Given two sets P and Q of pairwise-disjoint intervals on a line, where $|P| = n$ and $|Q| = O(n)$, determine whether there is a translation of P that makes it contained in Q.

All these problems are known to be 3SUM-hard. If $q = 0$, then the q-Area Triangle problem is the same as determining whether three points of S are collinear, which is known to be 3SUM-hard [19]. If we draw unit disks centered at the points of S, then the deepest region in this disk arrangement corresponds to a location for the center of the unit disk that would contain most points. Determining the deepest region in a disk arrangement[1] is known to be 3SUM-hard [3], which can be used to show the 3-SUM-hardness of q-Points in a Disk. Barequet and Har-Peled [5] showed that the Interval Containment problem is 3-SUM-hard.

While examining the Interval Containment problem, we noticed that our techniques generalize to a general Pair Search Problem: Given a problem P of size n, where a solution for P can be defined by a pair of elements in P, and a procedure A that can verify whether a given pair corresponds to a solution in $O(n^{1+o(1)})$ classical time, determine a solution pair for P. Consequently, we obtain $O(n^{1+o(1)})$-time quantum algorithms also for the following two problems which can be modeled using pair search.

Polygon Cutting: Given a simple n-vertex polygon P, an edge e of P and an integer $K > 2$, is there a line that intersects e and cuts the polygon into exactly K pieces?

Disjoint projections: Given a set S of n convex objects, determine a line such that the set objects project disjointly on that line.

The Polygon Cutting problem is known to be 3SUM-hard [24]. Disjoint projections can be solved in $O(n^2 \log n)$ time in classical computing model [16], but it is not yet known to be 3SUM-hard.

In the full version [22] of this paper we show how the pair search can be further generalized for d-tuple search or in $\mathbb{R}^d$, which is relevant for the kSUM hard problems [10, 14].

2 Preliminaries

In this section, we describe some standard quantum procedures and tools from the literature that we will utilize to design our algorithms.

Theorem 1 (Grover Search [20]) *Let $X = \{x_1, x_2, ..., x_n\}$ be a set of n elements and let $f : X \to \{0, 1\}$ a boolean function. There is a bounded-error quantum procedure that can find an element $x \in X$ such that $f(x) = 1$ using $O(\sqrt{n})$ quantum queries.*

Theorem 2 (Amplitude Amplification [6]) *Let A be a quantum procedure with a one-sided error and success probability of at least ϵ. Then there is a quantum procedure B that solves the same problem with a success probability $\frac{2}{3}$ invoking A for $O(\frac{1}{\sqrt{\epsilon}})$ times.*

By repeating Amplitude Amplification a constant number of times we can achieve a success probability of $1 - \epsilon$ for any $\epsilon > 0$. This technique has been widely used in the literature to speed up classical algorithms.

Algorithm 1 presents the Recursive-Quantum-Search (RQS) of Ambainis and Larka [2] for searching over a subdivision, but we slightly modify the description to present it in terms of subproblems. We first describe the idea and summarize it in a theorem (Theorem 3) so that it can be used as a black box. We then illustrate the concept using the Point-On-3-Lines problem.

The algorithm decomposes the problem into $O(k^2)$ subproblems, where k is a carefully chosen parameter, and then checks whether there is a solution that spans at least two subproblems but does not evaluate the subproblems. If no such solution exists, then the solution is determined by one of the subproblems. If all the subproblems are sufficiently small, then it searches for

[1] Although the reduction of [3] uses disks of various radii, it is straightforward to modify the proof with same size disks.

Algorithm 1 Recursive-Quantum-Search (RQS)

Procedure RQS(M, n, δ, ϵ), where M is a problem of size at most n, δ is a positive constant, and ϵ is an allowable error parameter.
if $|M| < k$, where $k = n^{1/\alpha} \cdot \delta(\log n + \log \epsilon^{-1})$ and $\alpha \in O(\sqrt{\log n / \log\log n})$, **then**
    Search for a solution by exhaustive search
else
    Let $R_1, \ldots, R_t$ be a decomposition of the problem M into t subproblems, where $t \in O(k^2)$. Search for a solution that spans two or more subproblems.
    if any of the subproblems is larger than $\frac{|M|}{k} \cdot \delta(\log |M| + \log \epsilon^{-1})$ **then**
        return Error
    else
        Let A be an algorithm that runs RQS (R, n, δ, ϵ) recursively on randomly chosen subproblem R. Run A with Amplitude Amplification for a success probability at least $1 - \epsilon$.

a solution over them using Grover search; otherwise, it returns an error. Consequently, one needs to show that the probability of a subproblem being large can be bounded by an allowable error parameter ϵ, and hence, Grover search will ensure a faster running time. An example application of this algorithm is provided later in this section.

We now have the following theorem, which is inspired by Ambainis and Larka's [2] result, but we include it here for completeness.

Theorem 3 *Let M be a problem of size at most n. Assume that for every $k < |M|$, M can be decomposed into $O(k^2)$ subproblems such that M can be solved first by checking for solutions that span at least two subproblems (without evaluating the subproblems), and then, if such a solution is not found, applying a Grover search over these subproblems (when we evaluate the subproblems). Furthermore, assume there exists a constant δ such that the probability for a subproblem to have a size larger than $\frac{|M|}{k} \cdot \delta(\log |M| + \log \epsilon^{-1})$ is at most ϵ, where ϵ is an allowable error probability.*

If we can compute the problem decomposition and check whether there is a solution that spans at least two subproblems in $O(|M|^{1+o(1)}k^2)$ classical time, then RQS can solve M in $O(n^{1+o(1)})$ quantum time.

Proof. The first time RQS is called, M is the original problem with size $|M| = n$. Since the recursion tree has a branching factor of $O(k^2)$, the number of problems at level j is $C_1 k^{2j}$, where C_1 is a constant.

We set k to be $n^{1/\alpha} \cdot \delta(\log n + \log \epsilon^{-1})$, where $\alpha \in O(\sqrt{\log n / \log\log n})$. At each recursion, the problem size decreases by a factor of $n^{-1/\alpha}$, and at jth level, a problem has size at most $n^{1-j/\alpha}$. Since the cost of problem decomposition and checking whether a solution spans two or more subproblems is $O(|M|^{1+o(1)}k^2)$, using Grover search, the cost for level j is $\sqrt{C_1 k^{2j}} \cdot C_2(n^{1-j/\alpha} n^{2/\alpha} n^{o(1)})$, where C_2 is a constant. We sum the cost of all levels to bound $T(n)$.

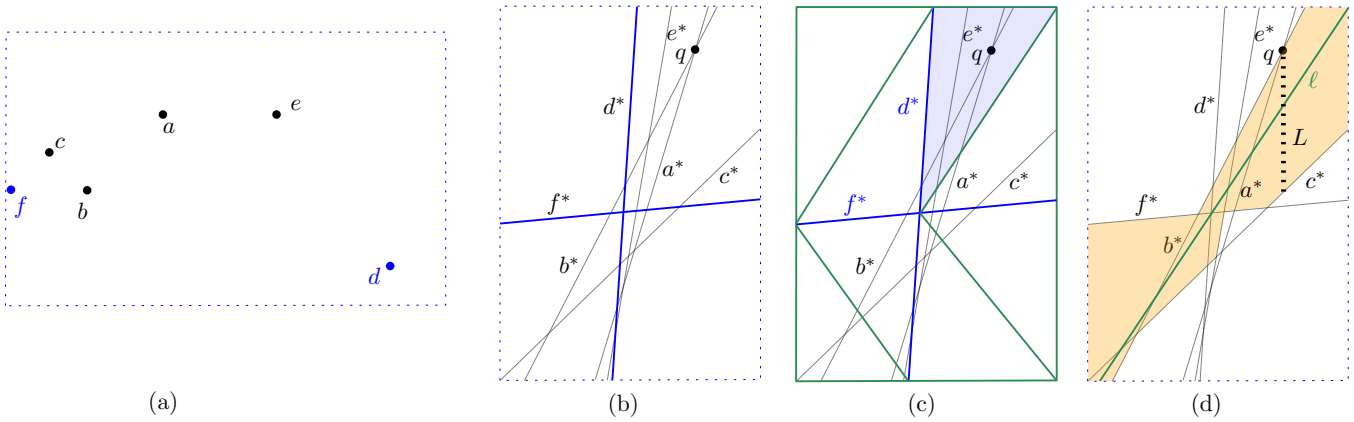
$$
\begin{aligned}
T(n) &\le C_2 \sum_{j=0}^{\alpha} \sqrt{(C_1 k)^{2j}} \left(n^{1-j/\alpha} n^{2/\alpha} n^{o(1)} \right) \\
&= C_2 n^{1+(2/\alpha)+o(1)} \sum_{j=0}^{\alpha} \left(\frac{C_1 k}{n^{1/\alpha}} \right)^j \\
&= C_2 n^{1+(2/\alpha)+o(1)} \sum_{j=0}^{\alpha} \left(\frac{C_1 n^{1/\alpha} \delta(\log n + \log \epsilon^{-1})}{n^{1/\alpha}} \right)^j \\
&\le C_2 n^{1+(2/\alpha)+o(1)} \sum_{j=0}^{\alpha} (C_3 \log n)^j \\
&\le C_2 \alpha (C_3 \log n)^{\alpha} n^{1+(2/\alpha)+o(1)} \\
&= C_2 \alpha \left(\frac{C_3 \log n}{n^{2/\alpha^2}} \right)^{\alpha} \left(n^{1+2/\alpha} n^{(2/\alpha)+o(1)} \right)
\end{aligned}
$$

If $\alpha = \sqrt{\frac{2\log(n)}{\log(C_3)+\log\log(n)}}$, then $n^{\frac{2}{\alpha^2}} = C_3 \log(n)$. Hence $T(n) = C_2 \alpha n^{1+\frac{4}{\alpha}+o(1)} = O(n^{1+o(1)})$.

□

An Inspiring Example: We can use Theorem 3 as a black box. For example, consider the case of Point-On-3-Lines problem. Let S be the set of input lines. To construct subproblems, choose k lines randomly, then create an arrangement of these lines, and finally, triangulate the arrangement to obtain $O(k^2)$ faces. Specifically, a subproblem corresponding to a closed face F consists of the input lines that bound F and the lines that intersect the interior of F. If a solution point (i.e., a common point on three lines) spans at least two closed faces, then it must lie on an edge or coincide with a vertex of the triangulation, which can be checked in $O(|S|k^2 \log n)$ time without evaluating the subproblems. If a solution point is not found, then we can search over the subproblems using Grover search. Ambainis and Larka's [2] showed that there exists a constant δ such that the probability of a subproblem to contain more than $\delta \frac{|S|}{k}(\log(|S|) + \log(\epsilon^{-1}))$ lines is bounded by ϵ, and hence, we can apply Theorem 3. The following lemma, which is adapted from [2], will be helpful for us to argue about subproblem sizes.

Lemma 4 (Ambainis and Larka [2]) *Let S be a set of straight lines and let A be an arrangement of k lines that are randomly chosen from S. Let $\mathcal{T}$ be a planar subdivision of size $O(k^2)$ obtained by adding straight line segments to A such that each face of $\mathcal{T}$ is of size $O(1)$.*



(a) (b) (c) (d)

Figure 1: (a)–(b) Illustration for a point set and its corresponding lines in the dual plane. (c) Illustration for T_k, where $k = 2$, with a face R shown in blue shaded region. The dual and supporting edges are in blue and green, respectively. (d) Illustration for the zone of a supporting line.

Then the probability of a closed face of T, without its vertices, intersecting more than $\delta\frac{|S|}{k}(\log(|S|)+\log(\epsilon^{-1}))$ lines of S is bounded by ϵ, where δ is a positive constant and ϵ is an allowable error probability.

The reason to restrict the attention to a closed face without its vertices (in Lemma 4) is to avoid the degenerate case with many lines intersecting at a common point. In such a scenario, a random sample of S is likely to have many lines passing through such a point, yielding a closed face intersected by many lines. Ambainis and Larka's proof [2] did not explicitly discuss this scenario.

3 Finding a Triangle of Area at most q

A well-known approach for finding a minimum area triangle among a set S of n points [15] is to use point-line duality. For each point $p = (p_x, p_y)$, construct a line p^*, which is defined as $y = p_x x - p_y$ in the dual plane. Figure 1(b) illustrates a set of dual lines corresponding to the points of Figure 1(a). Let o be the intersection of two dual lines a^* and b^*. The algorithm uses the property that the dual line c^* with the smallest vertical distance from o determines a triangle Δabc that minimizes the area over all the triangles that must include a and b. Here a vertical distance is defined by the length of the smallest line segment parallel to the y-axis connecting o and c^*. Consequently, one can first construct a line arrangement in the dual plane and then examine its faces to find a minimum area triangle in $O(n^2)$ classical time.

We now describe our approach using quantum computing. One can check whether three lines in the dual plane intersect at a common point in $O(n^{1+o(1)})$ quantum time [2], and if so, it would correspond to a triangle of 0 area. Therefore, we may assume that the lines are in a general position.

We first discuss the concept of 'zone' in an arrangement and some properties of a minimum area triangle. Let A_k be an arrangement of a set S_k^* of k randomly chosen dual lines, and let T_k be a triangulation obtained from A_k in $O(k^2)$ time. The *zone* of a line ℓ is the set of closed faces in A_k intersected by ℓ. Figure 1(b)–(c) illustrates a scenario where two lines d^*, f^* have been chosen to create T_k. For each face R in T_k, $s(R)$ denotes the dual lines (a subset of S_k^*) that bound R and the dual lines that intersect the interior of R. We refer to an edge of T_k as a *dual edge* if it corresponds to a dual line of a point in S, otherwise, we call it a *supporting edge*. The line determined by a supporting edge is called a *supporting line*. We now have the following property of a minimum area triangle.

Lemma 5 *Let Δabc be a minimum area triangle. Let q be the intersection point of a^* and b^*. Assume that q is not a vertex of T_k and q lies interior to a face R of T_k (e.g., Figure 1(c)). Then either one of the following or both hold: (a) c^* belongs to $s(R)$. (b) c^* belongs to $s(Z)$, where Z is a zone of a supporting line of R.*

Proof. Assume that (a) does not hold. We now show that (b) must be satisfied. Consider a vertical line segment L starting from q and ending on c^*. Since c^* minimizes the vertical distance from q, no other dual edge can intersect L (e.g., Figure 1(d)). Since q is enclosed by R and since c^* is outside of R, there must be a supporting edge ℓ on the boundary of R that intersects L. If the zone of the corresponding supporting line does not contain c^*, then we can find a dual line other than c^* that intersects L, which contradicts the optimality of Δabc. Figure 1(d) illustrates the zone of ℓ, which is shaded in orange. □

We now show how to leverage Theorem 3.

Let $R_1, \ldots, R_t$ be the faces of T_k. We choose $s(R_1), \ldots, s(R_t)$ as the subproblems. By Lemma 4, the probability of a subproblem being large is bounded by ϵ. In Lemma 6, we show how in $O(nk^2 \log n)$ time, one can check whether there is a triangle Δabc of area at most q such that no subproblem contains all three dual lines a^*, b^*, c^*. Consequently, we obtain Theorem 7.

Lemma 6 *A triangle that has an area of at most q and spans at least two subproblems can be computed in $O(nk^2 \log n)$ time.*

Proof. Each candidate triangle Δabc satisfies the property that the intersection point q of two of its dual lines lies in some face R and the third dual line does not intersect $s(R)$. Here the condition (b) of Lemma 5 must hold and it suffices to examine the zone of each supporting line of R. We thus check the zones of all the supporting lines of T_k as follows. Specifically, given an arbitrary line, its zone in an arrangement of n lines can be constructed in $O(n \log n)$ time [29]. Let e be a supporting line and let Z_e be its zone. We search over all the faces of Z_e to find a (vertex, edge) pair, i.e., (v, L), such that they lie on opposite sides of e and minimize the vertical distance from v to L. To process a face F we construct two arrays L_u and L_b. Here L_u (L_b) is an array obtained by sorting the vertices on the upper (lower) envelope of F using x-coordinates in $O(|F| \log |F|)$ time. Since F is convex, for each vertex q in L_u (L_b), we can use L_b (L_u) to find the dual line that has the lowest vertical distance from q in $O(\log |F|)$ time. Since the number of edges in a zone is $O(n)$ [12], the total time required for processing all the faces is at most $O(n \log n)$. For $O(k^2)$ supporting lines, the running time becomes $O(k^2 n \log n)$. □

Theorem 7 *Given a set S of n points, one can determine whether there is a triangle with area at most q in $O(n^{1+o(1)})$ quantum time.*

4 Finding a Unit Disk with at least q Points

Let S be a set of n points and consider a set $\mathcal{D}$ of n unit disks, where each disk is centered at a distinct point from S. Note that to solve q-Points in a Disk, it suffices to check whether there is a point r that hits at least q disks in $\mathcal{D}$. However, searching for r using Theorem 3 requires tackling some challenges. First, we need to create a problem decomposition, where the probability of obtaining a large subproblem is bounded by an allowable error probability. This requires creating a subdivision (possibly with curves) where the size of each region (corresponding to a subproblem) is $O(1)$. Second, we need to find a technique to check for solutions that span two or more subproblems.

Consider an arrangement A_k of k randomly chosen disks from $\mathcal{D}$. We first discuss how the regions of A_k can be further divided to create a subdivision A'_k where the size of each region is $O(1)$.

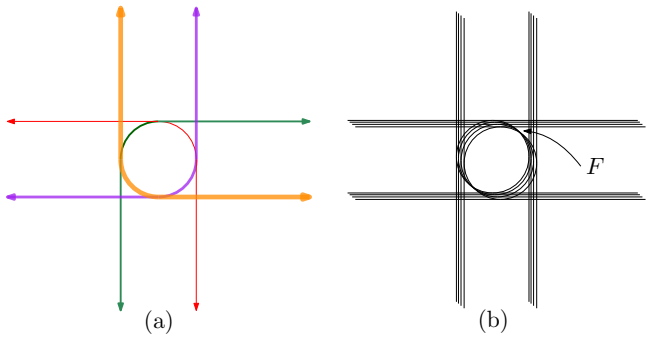


Figure 2: Illustration for the proof of Lemma 8.

Lemma 8 *Let A_k be an arrangement of k unit disks. In $O(k^2 \log n)$ time, one can create a subdivision of A'_k by adding straight line segments such that each face is of size $O(1)$.*

Proof. For each disk, we create four *pseudolines* as follows. Consider partitioning the disk into four regions by drawing a vertical and a horizontal line through its center. For each circular arc, we create a pseudoline by extending its endpoints by drawing two rays following the tangent lines, as shown in Figure 2(a). However, the resulting subdivision may still contain faces with linear complexity (e.g., the face F in Figure 2(b)). We subdivide each face further by extending a horizontal line segment from each vertex. The details are included in the full version [22]. At the end of the construction, each cell of the subdivision can be described using $O(1)$ arcs or segments. The construction inserts at most $O(k^2)$ straight lines and takes $O(\log n)$ time per addition to complete the process in $O(k^2 \log n)$ time. □

We now show how to leverage Theorem 3. Let $R_1, \ldots, R_t$ be the faces of A'_k. Let $s(R_i)$, where $1 \leq i \leq t$, be the disks that intersect the closed region R_i (except its vertices), but do not fully contain R. We subtract how many disks fully contain R from q and therefore, they should not be considered in the recursive subproblems. We show that the probability of a subproblem being large is bounded by ϵ (see the full version [22]). Lemma 9 shows how to check whether there is a solution point r (i.e., a point hitting at least q disks) that coincides with a vertex of A'_k or spans at least two subproblems in $O(nk^2 \log n)$ time. Consequently, we obtain Theorem 10.

Lemma 9 *Let r be a point that hits at least q disks. If r coincides with a vertex of A'_k or spans at least two subproblems then it can be found in $O(nk^2 \log n)$ time.*

Proof. For each edge $e = (v, w)$ of A'_k, we first count the number of disks intersected by v in $O(n)$ time. We

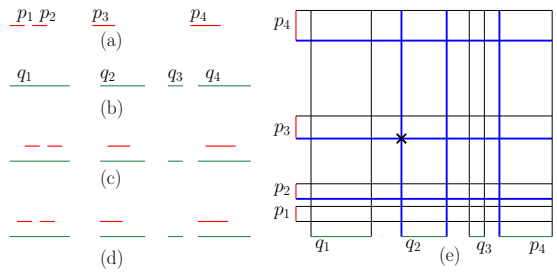


Figure 3: (a) P. (b) Q. (c) $P \subset Q$. Illustration for (d) Remark 11 and (e) Theorem 12.

then compute all the intersection points between e and the input disks and sort them based on their distances from v in $O(n \log n)$ time. Finally, we walk along e from v to w, and each time we hit an intersection point o, we update the current disk count (based on whether we are entering a new disk or exiting a current disk) to compute the number of disks intersected by o. □

Theorem 10 *Given a set S of n points, one can determine whether there is a unit disk with at least q points in $O(n^{1+o(1)})$ quantum time.*

5 Determining Interval Containment

Let $\mathcal{I}$ be an instace of Interval Containment, and let $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_m)$, where $m = O(n)$, be the two sets of pairwise disjoint intervals of $\mathcal{I}$. We now give an $O(n^{1+o(1)})$-time quantum algorithm to determine whether P can be translated so that it becomes contained in Q. If there is an affirmative solution, then we can continuously move the intervals in P until an endpoint of one of its intervals hits an endpoint of an interval of Q, as shown in Figure 3(c)–(d).

Remark 11 *If $\mathcal{I}$ admits an affirmative solution, then there is a solution where an end point of one interval of P coincides with an end point of an interval in Q.*

We now use Remark 11 to find a solution for $\mathcal{I}$.

Theorem 12 *Given two sets P and Q of $O(n)$ pairwise-disjoint intervals on a line, one can determine whether there is a translation of P that makes it contained in Q in $O(n^{1+o(1)})$ quantum time.*

Proof. We place the intervals of Q on the positive x-axis starting from $(1, 0)$ and the intervals of P on the positive y-axis starting from $(0, 1)$, as shown in Figures 3(e). Consider a set H of $2n$ horizontal lines and a set V of $2m$ vertical lines through the endpoints of the intervals of P and Q, respectively. Let A_k be an arrangement determined by k randomly chosen lines from $(H \cup V)$, e.g., the thick blue lines of Figures 3(e). Add the smallest area rectangle containing P and Q to the arrangement so that we get a subdivision T_k, where its faces $R_1, \ldots, R_t$ are rectangles. By $s(R_i)$, where $1 \leq i \leq t$, we denote the lines of H and V that intersect the closed region R_i.

We now show how to leverage Theorem 3. By Remark 11, it suffices to examine pairs of endpoints from P and Q. Let a be an endpoint from P and b be an endpoint from Q that determine a solution. Let o be the intersection point of the corresponding lines $\ell_a \in V$ and $\ell_b \in H$. We refer to o as the solution point, which may lie at a vertex, or interior to an edge, or interior to a face of T.

We choose $s(R_1), \ldots, s(R_t)$ as the subproblems. By Lemma 4, the probability of a subproblem being large is bounded by ϵ. In $O(nk^2 \log n)$ time, we can check whether o coincides with a vertex of T_k, i.e., spans at least two subproblems (Figure 3(e)). However, we do not check whether the solution o lies on an edge of T because if o lies interior to an edge or a face of T_k, then it is found by a Grover search over the subproblems. The running time follows directly from Theorem 3. □

6 Pair/Tuple Search and Generalizations

For a pair search problem P, if we can decide whether a given pair corresponds to a solution in $f(n) \in O(n^{o(1)})$ classical time, then a straightforward application to Grover search yields an $O(n^{1+o(1)})$-time quantum algorithm. However, we show how to solve P in $O(n^{1+o(1)})$-time even when $f(n) \in O(n^{1+o(1)})$.

Theorem 13 *Let P be a problem of size n where a solution for P can be defined by a pair of elements in P. Assume that we can decide whether a given pair corresponds to a solution in $O(n^{1+o(1)})$ classical time. Then a solution pair can be computed in $O(n^{1+o(1)})$ time using a quantum algorithm.*

Proof. We first label the elements of P from t_1 to t_n. For each element t_i, we create a horizontal line $y = i$ and a vertical line $x = i$. Every pair of lines (a, b), where one is horizontal and the other is vertical, corresponds to a pair of elements (t_a, t_b). Now the search over the subdivision is similar to the proof of Theorem 12. □

Theorem 13 allows for an $O(n^{1+o(1)})$-time quantum algorithm for Polygon Cutting and Disjoint projections problems. The details are in the full version [22]. The pair search technique can be applied to obtain quantum speed up as long as the check for a pair takes sub-quadratic time. For example, if a pair can be checked in $O(n^{1+\beta})$ classical time, then the analysis of Theorem 3 gives an algorithm with $O(n^{1+\beta})$ quantum time. Hence a maximum clique in a unit disk graph, where pairs of points are checked in $O(n^{1.5} \log n)$ classical time [17, 18], can be found in $O(n^{1.5})$ quantum

time. The pair search technique generalizes to d-tuple search, where one needs to search for a solution over an arrangement in $\mathbb{R}^d$. The full version [22] includes the details.

7 Conclusion

In this paper we discuss quantum speed-up for some geometric 3SUM-Hard problems. We also show how our technique can be applied to a more general pair or tuple search setting. A natural avenue to explore would be to establish nontrivial lower bounds under the Quantum Strong Exponential-Time Hypotheses [9] or quantum 3SUM conjecture [8].

8 Acknowledgements

This research work is supported in part by NSERC — Alliance International Catalyst Quantum grants. We thank Sharma V. Thankachan for encouraging discussions.

References

[1] S. Aaronson, N.-H. Chia, H.-H. Lin, C. Wang, and R. Zhang. On the quantum complexity of closest pair and related problems. In *Proceedings of the 35th Computational Complexity Conference*, 2020.

[2] A. Ambainis and N. Larka. Quantum algorithms for computational geometry problems, 2020.

[3] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.

[4] I. Baran, E. D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008.

[5] G. Barequet and S. Har-Peled. Polygon containment and translational min-Hausdorff-distance between segment sets are 3sum-hard. *Int. J. Comput. Geom. Appl.*, 11(4):465–474, 2001.

[6] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[7] H. Buhrman, B. Loff, S. Patro, and F. Speelman. Limits of quantum speed-ups for computational geometry and other problems: Fine-grained complexity via quantum walks. In *Proc. of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215 of *LIPIcs*, pages 31:1–31:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[8] H. Buhrman, B. Loff, S. Patro, and F. Speelman. Limits of quantum speed-ups for computational geometry and other problems: Fine-grained complexity via quantum walks. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[9] H. Buhrman, S. Patro, and F. Speelman. A framework of quantum strong exponential-time hypotheses. In *Proceedings of the 38th international symposium on theoretical aspects of computer science (STACS 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021.

[10] J. Cardinal, J. Iacono, and A. Ooms. Solving k-SUM using few linear queries. In P. Sankowski and C. D. Zaroliagis, editors, *Proceedings of the 24th Annual European Symposium on Algorithms (ESA)*, volume 57 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[11] T. M. Chan. More logarithmic-factor speedups for 3sum,(median,+)-convolution, and some geometric 3sum-hard problems. *ACM Transactions on Algorithms (TALG)*, 16(1):1–23, 2019.

[12] B. Chazelle, L. J. Guibas, and D.-T. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25(1):76–90, 1985.

[13] K. L. Clarkson. New applications of random sampling in computational geometry. *Discret. Comput. Geom.*, 2:195–222, 1987.

[14] M. Dietzfelbinger. Lower bounds for sorting of sums. *Theoretical Computer Science*, 66(2):137–155, 1989.

[15] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38(1):165–194, 1989.

[16] H. Edelsbrunner, M. Overmars, and D. Wood. Graphics in flatland: A case study. In *Computational Geometry: Theory and Applications*, volume 1. 1983.

[17] D. Eppstein. Graph-theoretic solutions to computational geometry problems. In C. Paul and M. Habib, editors, *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 1–16, 2009.

[18] J. Espenant, J. M. Keil, and D. Mondal. Finding a maximum clique in a disk graph. In E. W. Chambers and J. Gudmundsson, editors, *Proceedings of the 39th International Symposium on Computational Geometry (SoCG)*, volume 258 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[19] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 45(4):140–152, 2012.

[20] L. K. Grover. A framework for fast quantum mechanical algorithms. In *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 53–62. ACM, 1998.

[21] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. In *Proceedings of the Second Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry*, pages 61–71. ACM, 1986.

[22] J. M. Keil, F. McLeod, and D. Mondal. Quantum speedup for some geometric 3SUM-Hard problems and beyond. *arXiv preprint arXiv:2404.04535*, 2024.

[23] T. Lafaye and R. Kulkarni. Quantum query complexity in computational geometry revisited. In *Quantum Information and Computation IV*, volume 6244, pages 291–297. SPIE, 2006.

[24] E. Ruci. *Cutting a Polygon with a Line*. PhD thesis, Carleton University, 2008.

[25] K. Sadakane, N. Sugawara, and T. Tokuyama. Quantum algorithms for intersection and proximity problems. In *Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC)*, pages 148–159. Springer, 2001.

[26] K. Sadakane, N. Sugawara, and T. Tokuyama. Quantum computation in computational geometry. *Interdisciplinary information sciences*, 8(2):129–136, 2002.

[27] N. Volpato and A. Moura. Tight quantum bounds for computational geometry problems. *International Journal of Quantum Information*, 7(05):935–947, 2009.

[28] N. Volpato and A. Moura. A fast quantum algorithm for closest bichromatic pair problem. *Univ. Estad. Campinas, TR IC-10-03*, 2010.

[29] H. Wang. A simple algorithm for computing the zone of a line in an arrangement of lines. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 79–86. SIAM, 2022.

Experimental analysis of oriented spanners on one-dimensional point sets

Kevin Buchin* Antonia Kalb* Guangping Li* Carolin Rehs*

Abstract

Given a point set P in the Euclidean space and a parameter t, an *oriented t-spanner* G is an oriented subgraph of the complete bi-directed graph, such that for every pair of points, the shortest closed walk in G containing those points is at most a factor t longer than their shortest cycle in the complete undirected graph on P. Since it is known that minimising the oriented dilation for a set of points in the plane is NP-hard, we focus on one-dimensional point sets. Previous work has studied oriented spanners that have a book embedding on the given point set, but neither the minimum-achievable dilation nor the structure of minimum (oriented) dilation spanners is well-understood.

We present a formulation of the problem of computing such minimum dilation spanners as satisfiability problem. We utilise this formulation for computational experiments on these spanners, resulting in several interesting insights. Firstly, we find point sets such that the minimum dilation spanners with a one-page book embedding on the point sets have an oriented dilation arbitrarily close to $1+\Phi$, where $\Phi \approx 1.618$ is the golden ratio, improving upon the previously known lower bound on the worst-case dilation of 2. Secondly, for spanners with a two-page book embedding and under the assumption that edges between consecutive numbers are all oriented from the smaller to the larger, we find a point set with minimum oriented dilation $\sqrt{2}$. We make further structural observations in this setting.

1 Introduction

Geometric spanners are a well-studied problem for decades (see [4, 9] for surveys) and admit many applications. Recently, this concept has been extended to oriented spanners [5], motivated by applications that require one-way edges, as for low-interference ad-hoc networks, motion planning, one-way road networks or non-bidirectional communication networks.

Given a point set P in the Euclidean space and an oriented graph G on P, the *oriented dilation of two points* $p, p' \in P$ is defined as $\mathrm{odil}_G(p,p') = \frac{|C_G(p,p')|}{|\Delta(p,p')|}$, where $C_G(p,p')$ denotes the *shortest closed walk* in G containing the points p and p' and $\Delta(p,p')$ is the shortest cycle containing the points p and p' in the complete undirected graph on P. The oriented dilation t of G is defined as $t = \max_{p,p' \in P} \mathrm{odil}_G(p,p')$. A spanner is called a *minimum (oriented) spanner* if it minimises the oriented dilation t.

As minimising the oriented dilation is NP-hard for point sets in the Euclidean plane, previous algorithmic work has focused on special cases and on one-dimensional point sets [5].

On one-dimensional point sets in particular spanners with a book embedding were studied, as natural one-dimensional analogue to plane spanners in two dimensions.

A *one-page book embedding* [3, 6, 7] of a graph corresponds to an embedding of the vertices as points on a horizontal line with the edges drawn without crossings as circular arcs above the line; and also below in a *two-page book embedding*. In such a (one- or two-page) book embedding, for consecutive points on the line, we may draw their edge straight on the line.

For one-dimensional point sets, there is a polynomial-time algorithm to compute a minimum oriented spanner with a one-page book embedding. Its worst-case dilation is bounded by $2 \leq t \leq 5$. Further, there exists a simple construction for a 2-spanner with a two-page book embedding [5]. For both one-page and two-page book embeddings this leaves open the intriguing question, how large the minimum oriented dilation is in the worst-case. For two-page book embeddings additionally the question remains open how to compute a minimum-dilation spanner.

The aim of this paper is to address these questions through computational experiments. For this, we first analyse book embeddings more carefully and model the decision problem of finding a minimum t-spanner for a given point set as Satisfiability (SAT) formula in Section 3.

Utilising this formulation, we perform experiments on random point sets. For one-page book embeddings, we observe in Section 4 that the minimum dilation stays below $1+\Phi$, where Φ is the golden ratio. By inspecting examples with large dilation, we found a family of point sets of size 6 with minimum dilation arbitrarily close to $1+\Phi$ (from below).

For two-page book embeddings, we focus on spanners with a so called "baseline", that is, all edges between

*Technical University of Dortmund, Germany, {kevin.buchin,guangping.li,antonia.kalb, carolin.rehs}@tu-dortmund.de

consecutive points (in the one-dimensional order) have an edge oriented from the point with the smaller coordinate to the point with the larger coordinate. In Section 5 we observe that in this setting the minimum oriented dilation stays below $\sqrt{2}$. We provide a point set of size 6 with minimum dilation $\sqrt{2}$.

In the setting of one-page book embeddings, it is known that arbitrary *long* edges may be needed to minimise the dilation, where the *length* of an edge counts how many points lie in between the end points (plus one); see Section 2 for a formal definition. Our experiments indicate that the situation is drastically different for two-page book embeddings with a baseline, where the maximum length we observed is 5. If true in general, this would pave the way to a polynomial-time algorithm for computing minimum spanners in this setting.

2 Preliminaries

Definitions In this paper, one-dimensional point sets are indexed in increasing order, i.e. p_1 is the leftmost point and p_n the rightmost point.

The *length* of an edge (p_i, p_j) is the Euclidean distance $|p_i - p_j|$. We call $|i - j|$ the *edge-distance* of (p_i, p_j).

A *walk* is defined as a sequence of points and edges of a graph. The *length of a walk* is the sum of the lengths of its edges. A walk is called *closed* if it starts and ends at the same point. A *path* is a walk where all points and edges are distinct.

We call a graph *one-page plane* (respectively *two-page plane*) if it has a one-page (resp. two-page) book embedding. A *maximal* one-page plane (analogous *two-page plane*) graph is a one-page plane graph $G = (P, E)$ such that for every edge $e \notin E$, the graph $G' = (P, E \cup \{e\})$ is not one-page plane.

A forward edge (p_i, p_{i+1}) between consecutive points is called a *baseline edge* and a backwards edge (p_j, p_i) with $i < j$ a *back edge*. A natural restriction of oriented graphs for one-dimensional point sets is to include a *baseline* and all other edges are back edges. We call such a graph a *one-page-plane-baseline graph*, for short 1-*PPB graph*. Analogously we define 2-*PPB graphs*. We note that for one-page plane graphs this restriction does not change the minimum dilation attainable [5], i.e., there is always a minimum spanner with this property. There exists a point set where this is not the case for two-page plane graphs.

Oriented dilation The following lemmas simplify the computation of the dilation of oriented spanners for one-dimensional point sets:

Lemma 1 [5] *Let P be a one-dimensional point set of n points. The oriented dilation t of an oriented graph G on P is $t = \max_{\substack{1\le i\le n-2\\ 1\le j\le n-3}} \{\mathrm{odil}_G(p_i, p_{i+2}), \mathrm{odil}_G(p_j, p_{j+3})\}$.*

This holds for every graph on a one-dimensional point set, even if the graph is neither plane nor maximal.

Lemma 2 [5] *Let P be a one-dimensional point set of n points. The oriented dilation t of a 1-PPB graph for P is $t = \max_{1\le i\le n-2} \mathrm{odil}_G(p_i, p_{i+2})$.*

Since Lemma 1 directly leads to a 1-spanner with a 3-page book embedding (i.e. a crossing-free embedding using a third half-plane) for every one-dimensional point set, only 1- and 2-page book embeddings are interesting to study. Obviously, for any point set P, there is no one-page plane 1-spanner if $|P| > 3$, and no two-page plane 1-spanner if $|P| > 4$.

Experimental Setup The experiments were run on a server equipped with two Intel Xeon E5-2640 v4 processors (2.4 GHz 20-core) and 64GB RAM. The code was compiled with g++ 13.1.0 with optimisation level -O3. Source code and benchmark data generator are at tudgpl.github.io/Oriented_Spanners. To solve our SAT formulation (Section 3), we use the solver Glucose 4.2.1 [1, 2], which is a freely available solver which ranks highly in the competition held at the annual SAT competition. Glucose is based on the solver Minisat 2.2 [8].

We generated synthetic data sets for our experiments. The synthetic data sets are randomly generated one-dimensional point sets of integers inside the range of 10^5 pixels, based on a uniform distribution.

Due to the limited computation resources, for the exploration of lower bounds, we start our experiments with small point sets of size n in $\{5, \ldots, 10, 16, 32\}$. Note that our theoretical lower bounds are reached with only six points. For each size, we generate 10^6 instances.

For the exploration of minimum 2-PPB spanners with edge-distance constraints, we vary the number of points from 16 to 512, with logarithmic increments. For each size, we generate 100 instances.

Our results are shown as a boxplot, which presents the distribution of quantitative data. It shows quartiles in a box and extends whiskers to illustrate the rest of the data, excluding outliers as separate dots.

3 Exact solver for 1- and 2-PPB graphs

In this section, we present a SAT model based solver to compute optimal 1-PPB or 2-PPB graphs.

Let us first consider shortest closed walks of point pairs in a 1- or 2-PPB graph:

Lemma 3 *Let P be a one-dimensional point set and $G = (P, E)$ a 1- or 2-PPB graph. Let p_i, p_j be points in P with $j \in \{i + 2, i + 3\}$. Let p_l with $l \le i$ be leftmost point in a shortest closed walk $C_G(p_i, p_j)$ and p_r with $j \le r$ is the rightmost point in $C_G(p_i, p_j)$. The closed walk $C_G(p_i, p_j)$ consists of the baseline edges from p_l to*

p_r, *i.e.* $\{(p_m, p_{m+1}) \mid l \leq m \leq r-1\}$, *and a path* Π *from* p_r *to* p_l *with either*

(a) $\Pi = (p_r, p_l)$ *(see Fig. 1a),*

(b) $\Pi = (p_r, p_k), (p_k, p_l)$ *with* $i < k < j$ *(see Fig. 1b), or*

(c) $\Pi = (p_r, p_{i+1}), (p_{i+1}, p_{i+2}), (p_{i+2}, p_l)$ *(see Fig. 1c).*

Note that, due to planarity, case c arises only if G *is a* 2*-PPB graph and* $j = i + 3$.

Proof. By definition, every 1- or 2-PPB graph includes a baseline. Therefore, the baseline contains a shortest path from p_l to p_r. Next, to complete the shortest closed walk $C_G(p_i, p_j)$, we consider the shortest path Π from p_r to p_l.

If $(p_r, p_l) \in E$, it holds $\Pi = (p_r, p_l)$, thus case a.

If $(p_r, p_l) \notin E$, the path Π is a union of back edges and maybe also baseline edges. We consider the first edge of Π. Since p_r is the rightmost point in $C_G(p_i, p_j)$, this edge must be a back edge (p_r, p_k) and therefore $k < r$. Note that $k > i$, otherwise, (p_r, p_k) is a path that covers p_i and p_j and $p_r - p_k < p_r - p_l$ which contradicts the optimality of $C_G(p_i, p_j)$. Furthermore, $k < j$, otherwise, the subpath of Π connecting p_k to p_l covers p_i and p_j and has length $p_k - p_l < p_r - p_l$ which contradicts the optimality of $C_G(p_i, p_j)$.

If $j = i + 2$ (and therefore $k = i + 1$), the second edge in Π must be a back edge $(p_k, p_{k'})$ with $k' < i$. Otherwise, if the second edge is (p_{i+1}, p_{i+2}), the subpath of Π connecting p_{i+2} to p_l covers p_i and p_j and has length $p_j - p_l < p_r - p_l$ which contradicts the optimality of $C_G(p_i, p_j)$. Therefore, the back edge $(p_k, p_{k'})$ is the second edge in Π. Since p_l is the leftmost point in $C_G(p_i, p_j)$, this implies $k' = l$ and therefore $\Pi = (p_r, p_{i+1}), (p_{i+1}, p_l)$, thus case b.

If $j = i + 3$ and $k = i + 2$, analogous, the second edge in Π must be a back edge $(p_k, p_{k'})$. Since $k = i + 2$ implies $k' \leq i$, it holds $k' = l$ and therefore $\Pi = (p_r, p_{i+2}), (p_{i+2}, p_l)$, thus case b.

For $j = i + 3$ and $k = i + 1$, if the second edge in Π is a back edge $(p_k, p_{k'})$ with $k' \leq i$, it holds $\Pi = (p_r, p_{i+1}), (p_{i+1}, p_l)$, thus case b. Otherwise, if the second edge is (p_{i+1}, p_{i+2}), it holds $\Pi = (p_r, p_{i+1}), (p_{i+1}, p_{i+2}), (p_{i+2}, p_l)$, thus case c. □

Due to Lemmas 1, 2 and 3, the dilation of a minimum spanner for a given point set P is contained in the set

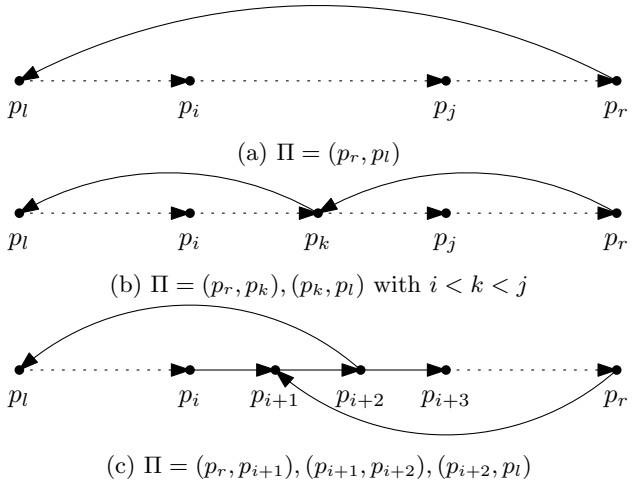


Figure 1: $C_G(p_i, p_j)$ consists of the baseline from p_l to p_r and a path Π from p_r to p_l with $j \in \{i + 2, i + 3\}$ and $l \leq i < j \leq r$

$\mathcal{T} \subset \mathbb{R}^+$ with

$$\mathcal{T} = \bigcup_{\substack{1 \leq i \leq n-2 \\ 1 \leq l \leq i \\ i+2 \leq r \leq n}} \left\{ \frac{p_r - p_l}{p_{i+2} - p_i} \right\} \cup \underbrace{\bigcup_{\substack{1 \leq i \leq n-3 \\ 1 \leq l \leq i \\ i+3 \leq r \leq n}} \left\{ \frac{p_r - p_l}{p_{i+3} - p_i}, \frac{p_r - p_l \ + \ p_{i+2} - p_{i+1}}{p_{i+3} - p_i} \right\}}_{\text{only for 2-PPB graphs}}.$$

So, there are $|\mathcal{T}| = O(n^3)$ possible values for the optimal dilation. Buchin et al. [5] bounded the oriented dilation of any 1-PPB spanner by 5 and any 2-PPB spanner by 2. Therefore, we first order $\mathcal{T}$ and do a binary search for each value $t \in \mathcal{T}$ in the range $[1, 5]$ for 1-PPB graphs and $[1, 2]$ for 2-PPB graphs.

Thus, to obtain the minimal dilation for some point set P, we consider the following decision for each candidate dilation t: Given a one-dimensional point set P and a parameter t, is there a 1-PPB (respectively 2-PPB) graph which is a t-spanner for P?

Now, we introduce our SAT model to solve this decision problem.

Variables. For every pair i, j with $j > i + 1$, there is a variable $e_{ij\varphi}$ with $\varphi \in \{1, 2\}$ which is true if and only if the solution set contains the back edge (p_j, p_i) on page φ.

For every pair l, r with $l + 1 < r$, due to Lemma 3, we are interested in only three possibilities for paths with p_l as leftmost and p_r as rightmost point. For each path possibility, we define a variable which is true if and only the corresponding path exists:

(a) $\Pi_a(l, r)$ represents the edge (p_r, p_l).

(b) $\Pi_b(l,k,r)$ represents the path $(p_r,p_k),(p_k,p_l)$ for $l+1<k<r-1$.

(c) For 2-PPB graphs: $\Pi_c(l,k,r)$ represents the path $(p_r,p_{k-1}),(p_{k-1},p_k),(p_k,p_l)$ for $l+2<k<r-1$.

Clauses. For every pair i,j, we list all the possible paths from p_j to p_i with length bounded by $t\cdot|p_j-p_i|$. Note that for 1-PPB graphs it suffices to consider only point pairs p_i,p_{i+2} (Lemma 2), and for 2-PPB graphs only pairs p_i,p_{i+2} and p_i,p_{i+3} (Lemma 1).

If a 1- or 2-PPB graph is maximal, for Lemma 3, case a holds either $l=i$, $r=j$ or both (compare to Fig. 1a). Since there is a minimum spanner with this property, this reduces the combinations that have to be considered for $\Pi_a(l,r)$.

Formally, for every pair i,j with $j=i+2$ for 1-PPB graphs and with $j\in\{i+2,i+3\}$ for 2-PPB graphs, we define:

(a) $\mathcal{P}_a(i,j)=\{\Pi_a(i,j)\}\vee\left\{\Pi_a(l,j)\mid\frac{p_j-p_l}{p_j-p_i}\le t\right\}\vee\left\{\Pi_a(i,r)\mid\frac{p_r-p_i}{p_j-p_i}\le t\right\}$,

(b) $\mathcal{P}_b(i,j)=\left\{\Pi_b(l,i+1,r)\mid\frac{p_r-p_l}{p_j-p_i}\le t\right\}\vee\left\{\Pi_b(l,i+2,r)\mid\frac{p_r-p_l}{p_j-p_i}\le t\right\}$, and

(c) For 2-PPB graphs: $\mathcal{P}_c(i,j)=\left\{\Pi_c(l,i+2,r)\mid\frac{p_r-p_l+p_{i+2}-p_{i+1}}{p_{i+3}-p_i}\le t\right\}$.

Finally, for every pair i,j with $j=i+2$ for 1-PPB graphs and with $j\in\{i+2,i+3\}$ for 2-PPB graphs, we add the following clause to the SAT formula:

- Bounded Dilation: $C_{ij}=\mathcal{P}_a(i,j)\cup\mathcal{P}_b(i,j)\cup\mathcal{P}_c(i,j)$

For each introduced path variable Π_x with $x\in\{a,b,c\}$, we add the following clause:

- Path-Constraint: $C_x=\{\Pi_x\}\cup\{\neg e_{mq1}\vee\neg e_{mq2}\mid(p_m,p_q)\in\Pi_x\}$

Finally, we add the following constraints for every edge (p_j,p_i) which occurs in our SAT formula:

- No-Repetition: $\neg e_{ij1}\vee\neg e_{ij2}$, and
- Planarity: $\neg e_{ij\varphi}\vee\neg e_{lk\varphi}$ and $\neg e_{ij\varphi}\vee\neg e_{kr\varphi}$ with $\varphi\in\{1,2\}$ and $l<i<k<j<r$.

4 One-Page Plane Spanners

In previous work [5], an upper lower bound of 2 for the dilation of one-page plane spanners is shown.

In the following, we present an instance of 6 points, which improves the lower bound for the minimum dilation of one-page plane spanners to 2.618.

Theorem 4 *There are one-dimensional point sets where no one-page plane t-spanner exists for $t<1+\Phi-\delta$, where Φ is the golden ratio $\Phi=1.618033\ldots$ and $\delta>0$ is an arbitrarily small value.*

Proof. Given $\delta>0$ we define a point set P such that for every 1-PPB t-spanner for P holds $t\ge1+\Phi-\delta$. Let P be a set of 6 one-dimensional points with the following pairwise distances: $p_2-p_1=p_6-p_5=\Phi$, $p_3-p_2=p_5-p_4=\epsilon$ and $p_4-p_3=1$ for a small $\epsilon>0$ (depending on δ).

It suffices to look at all maximal 1-PPB graphs for P. We enumerate all maximal 1-PPB graphs for P and, since P is symmetric, prune out every graph, which has an axial symmetric copy. Computing the dilation of every remained graph (listed in Appendix Fig. 5), we conclude that a minimum 1-PPB spanner for P has dilation $t=1+\frac{\Phi+\epsilon}{1+\epsilon}<1+\Phi$. For an arbitrary small $\epsilon>0$, this value approaches the bound arbitrarily closely. □

Computing optimal 1-PPB spanners. It suffices to look at maximal 1-PPB graphs to upper bound the minimum dilation of one-page plane spanners in general [5].

We solve our SAT formulation for 1-PPB graphs (Section 3) for uniform randomly generated point sets. Due to the limited computation resource, we started our experiments with small point sets. Note that a lower bound of 2.618, Theorem 4, is reached with only 6 points. (Experimental setup is given in section 2.)

Figure 2 shows the results of our experiments. Since the dilations are rounded down to three decimal places, a dilation of 1 seems to occur in the experiments. However, there is no one-page plane 1-spanner for $n>3$. The maximum and average dilation is listed in Table 1. As the number n of points grows, we see that the average dilation increases and the variance shrinks. A possible explanation for this could be that the maximum dilation requires specific local configurations, which become more likely when the number of points is increased. The maximum dilation for $n=6$ is significant larger than the maximum dilation for $n=5$. The maximum dilation of the minimum spanner for point sets of sizes $n\ge6$ is very similar, and stays slightly below $1+\Phi\approx2.618$.

Based on this, we conjecture that $1+\Phi$ is a general upper bound on the dilation of the minimum spanner. This would in particular mean that increasing the number of points beyond 6, does not result in a larger maximum dilation.

5 Two-Page Plane Spanners

Trivially, the dilation of two-page plane spanners is lower bounded by 1. An upper lower bound for this value has not been considered yet.

n	5	6	7	8	9	10	16	32
max	2.0	2.539	2.538	2.518	2.522	2.569	2.593	2.598
avg	1.463	1.591	1.668	1.732	1.779	1.816	1.943	2.059

Table 1: Maximum and average oriented dilation of minimum 1-PPB spanners on uniform randomly generated point sets of size n.

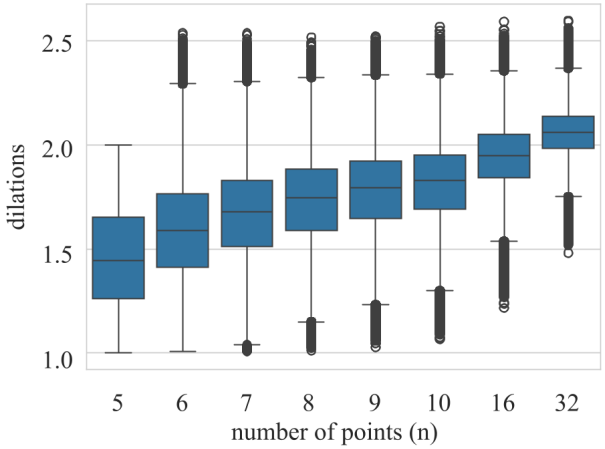


Figure 2: Boxplot of the dilation of minimum 1-PPB spanners. The x-axis shows the number of random points, the y-axis shows the dilation.

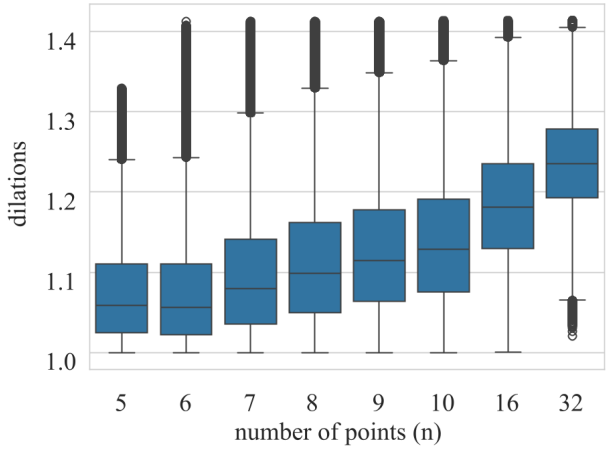


Figure 3: Boxplot of the dilation of minimum 2-PPB spanners. The x-axis shows the number of random points, the y-axis shows the dilation.

Analogous to Section 4, we present an instance of 6 points, which bounds the minimum dilation of 2-PPB spanners by 1.414.

Theorem 5 *There are one-dimensional point sets where no 2-PPB t-spanner exists for $t < \sqrt{2}$.*

Proof. Let P be a set of 6 one-dimensional points with the following pairwise distances: $p_2 - p_1 = p_4 - p_3 = p_6 - p_5 = 1$ and $p_3 - p_2 = p_5 - p_4 = \frac{1}{\sqrt{2}}$. We will show that for every 2-PPB t-spanner for P holds $t \geq \sqrt{2}$.

Since it suffices to consider only maximal 2-PPB spanners for P, we bruteforce all these spanners. We prune out every graph $G = (P, E)$ with $(p_{j'}, p_j), (p_j, p_i), (p_{j'}, p_i) \in E$ for $i + 1 < j < j' - 1$ and either $i \neq 1$ or $j' \neq 6$, because G has always the same dilation as $G' = (P, E \setminus \{(p_{j'}, p_i)\})$. Further, taking into account the symmetry of P, we prune out point symmetric duplicates. (All 2-PPB graphs for P are shown in Appendix Figure 6.) Computing the dilation of every remained graph, we conclude that a minimum 2-PPB spanner for P has dilation $t = \sqrt{2}$. □

Computing optimal 2-PPB spanners. We solve our SAT formulation for 2-PPB graphs (Section 3) for uniform randomly generated point sets. Since our lower bound of 1.414 is reached with only 6 points (Theorem 5), we start our experiments with small point sets. (The experimental setup is given in Section 2.)

Figure 3 shows the results of our experiments. Every dilation is rounded down to three decimals; in particular the dilation is always strictly larger than 1, since there is no two-page plane 1-spanner for $n > 4$. The maximum and average dilation is listed in Table 2. As in the experiments for one-page plane graphs, the average dilation increases and the variants of dilation shrink with a growing number of points. The maximum dilation of a minimum 2-PPB spanner varies between 1.329 for $n = 5$ and 1.413 for $n = 32$, which is below, but very close to the bound of $\sqrt{2} \approx 1.414$.

As for one-page plane spanners, the results of the experiments suggest that $\sqrt{2}$ is also an upper bound on the minimum dilation.

Bounded edge-distances For one-page plane spanners, there are point sets, where no minimum spanner has a constant edge-distance. (An example is the point set with pairwise distances $p_i - p_{i-1} = 2^i$.) For every considered point set, we computed a minimum 2-PPB spanner with edge-distance of at most 5. Therefore, in the last experiment, we explore the edge-distances in 2-PPB spanners. We compare the minimum spanner to the minimum spanner with bounded edge-distance.

n	5	6	7	8	9	10	16	32
max	1.329	1.412	1.412	1.412	1.412	1.413	1.413	1.413
avg	1.076	1.076	1.096	1.112	1.126	1.138	1.184	1.236

Table 2: Maximum and average oriented dilation of minimum 2-PPB spanners on uniform randomly generated point sets of size n.

	16	32	64	128	256	512
3	2%	1%	0%	0%	0%	0%
4	92%	95%	92%	90%	91%	91%
5	100%	100%	100%	100%	100%	100%

Table 3: Percentage of instances where a minimum spanner with edge-distance constraint has the same dilation than a minimum spanner without constraints. The row indices are the maximal allowed edge-distances and the column indices are for the number of random points.

For each $d \in [3, 20]$, we computed for each generated random instance a minimum spanner where for every back edge (p_j, p_i) it holds $j - i \leq d$ and a minimum spanner without constraints. Due to the limited computation resource, our extract solver only provides results for 91 (of 100) largest instances of 512 points.

We first consider the optimality ratios: The percentage of instances where a minimum spanner with only short edge-distance edges has the same dilation as a minimum spanner without constraints. The ratios are shown in Table 3. In our experiment, for each of the solved instances, a minimum spanner exists where the edge-distance of each edge is bounded by 5.

We proceed to check the qualities of spanners with only short edge-distance edges. Here we explain the result of instances of 64 points (see Fig. 4). The results for other point set sizes are similar (see Appendix Fig. 7). It is shown that from an edge-distance of 4, the range of dilations is quite similar, and from an edge-distance of 5 it is not changing anymore.

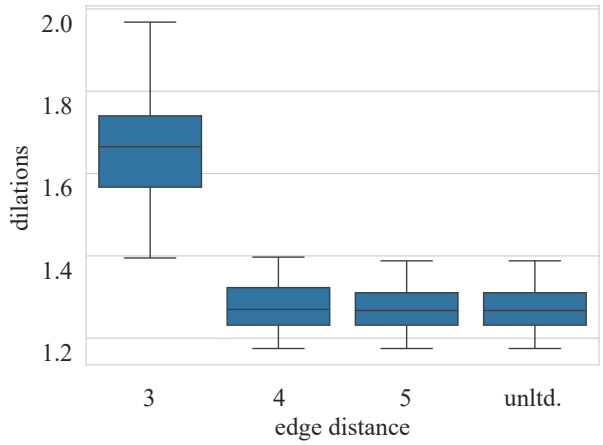


Figure 4: Boxplot of the dilation of minimum 2-PPB spanners with edge-distance constraints on point set of 64 points. The x-labels indicate the allowed longest edge-distance in the solutions, "unltd." is for minimum 2-PPB spanners without edge-distance constraints.

6 Conclusion

In this work, we experimentally explored the minimum dilation of oriented spanners on one-dimensional point sets. Specifically, we studied the worst-case (i.e. supremum of) the minimum dilation taken over all point sets.

The minimum dilation in the experiments stayed below $1 + \Phi$ and $\sqrt{2}$ for one- and two-page plane oriented spanners, respectively, where Φ is the golden ratio. Interestingly, the worst-case seems to already occur for point sets of size 6.

The experiments provide new lower bounds on the worst case minimum dilation. The current gap for the worst-case minimum dilation t is $1 + \Phi \leq t \leq 5$ for minimum one-page plane spanners, where Φ is the golden ratio, and $\sqrt{2} \leq t \leq 2$ for minimum 2-PPB spanners. Closing these gaps is an open problem. Based on the experiments we conjecture that that the lower bounds are tight.

The experiments suggest that there are minimum 2-PPB spanners only using "short" back edges, i.e., back edges (p_j, p_i) with $|j - i| \leq 5$ (edge-distance bounded by 5). If this property holds in general, it could be used as a base of a polynomial time algorithm, computing the spanner from left to right by a suitable dynamic program. Thus, following these experimental results, we aim to prove this property in future work.

References

[1] G. Audemard and L. Simon. Glucose: a solver that predicts learnt clauses quality. 01 2009.

[2] G. Audemard and L. Simon. On the glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018.

[3] M. A. Bekos, M. Gronemann, and C. N. Raftopoulou. Two-page book embeddings of 4-planar graphs. *Algorithmica*, 75(1):158–185, 2016.

[4] P. Bose and M. Smid. On plane geometric spanners: A survey and open problems. *Comput. Geom. Theory Appl.*, 46(7):818–830, 2013.

[5] K. Buchin, J. Gudmundsson, A. Kalb, A. Popov, C. Rehs, A. van Renssen, and S. Wong. Oriented spanners. In I. L. Gørtz, M. Farach-Colton, S. J. Puglisi, and G. Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 26:1–26:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

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

[7] V. Dujmovic and D. R. Wood. On linear layouts of graphs. *Discret. Math. Theor. Comput. Sci.*, 6(2):339–358, 2004.

[8] N. Eén and N. Sörensson. An extensible sat-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[9] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

Appendix

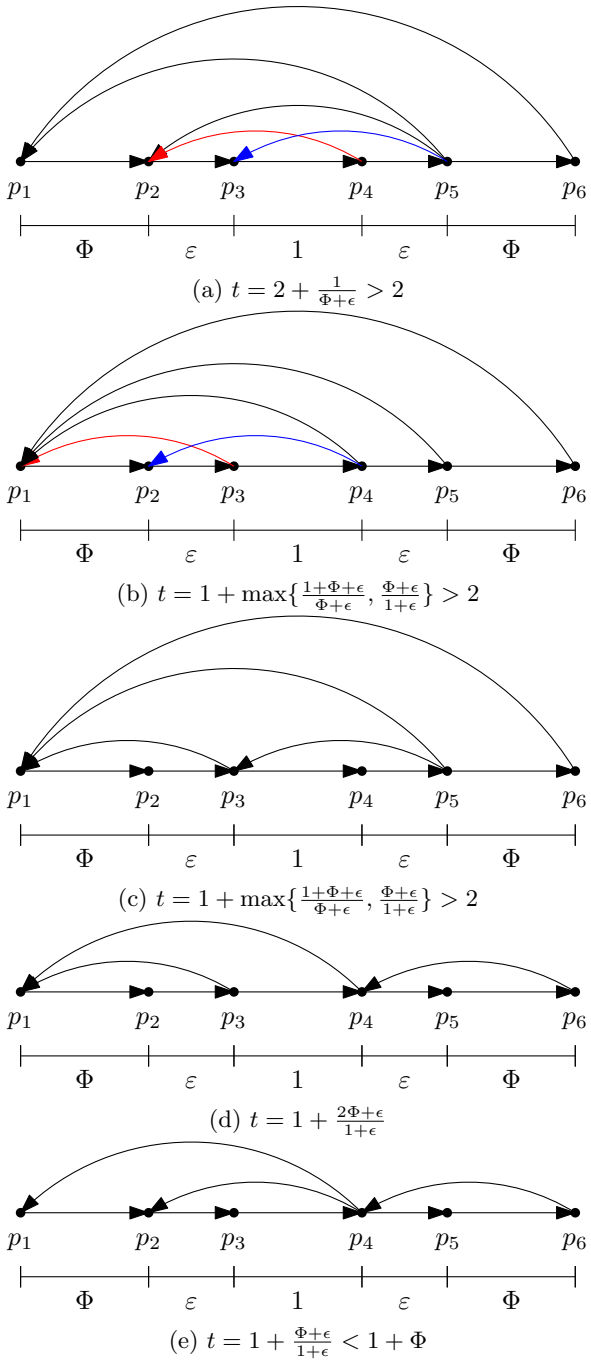
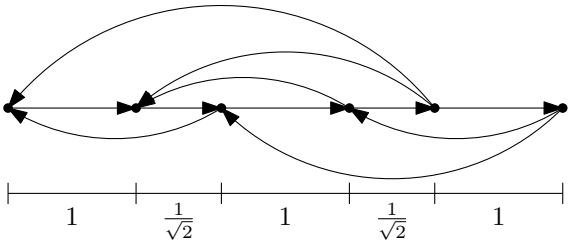
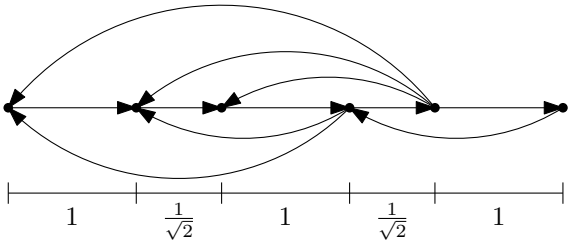


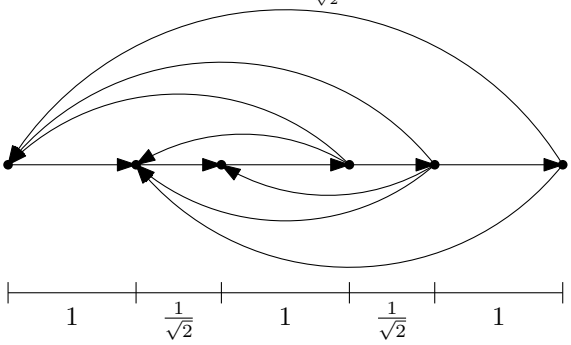
Figure 5: All 1-PPB graphs for P with $p_2 - p_1 = p_6 - p_5 = \Phi$, $p_3 - p_2 = p_5 - p_4 = \epsilon$ and $p_4 - p_3 = 1$, taking account to the symmetry of the point set, and their dilation t. Graphs with the same dilation are drawn in one subfigure with different coloured edges.



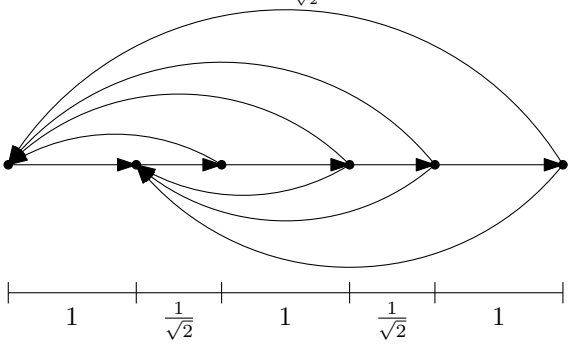
(a) $t = 1 + \frac{\frac{1}{\sqrt{2}}}{1+\frac{1}{\sqrt{2}}} = \sqrt{2}$



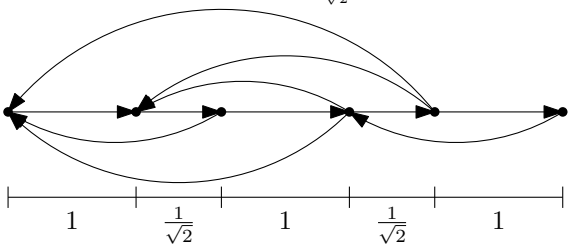
(b) $t = 1 + \frac{1}{1+\frac{1}{\sqrt{2}}} = 3 - \sqrt{2}$



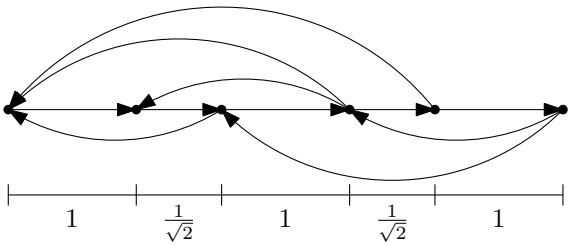
(c) $t = 1 + \frac{1+\frac{1}{\sqrt{2}}}{1+\frac{1}{\sqrt{2}}} = 2$



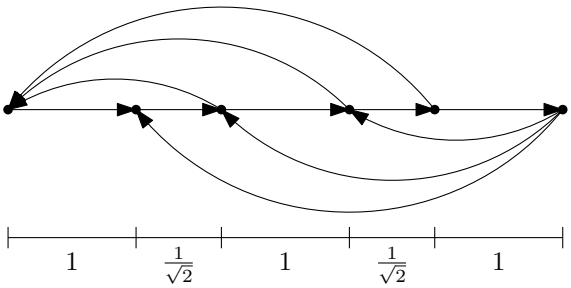
(d) $t = 1 + \frac{1+\frac{1}{\sqrt{2}}}{1+\frac{1}{\sqrt{2}}} = 2$



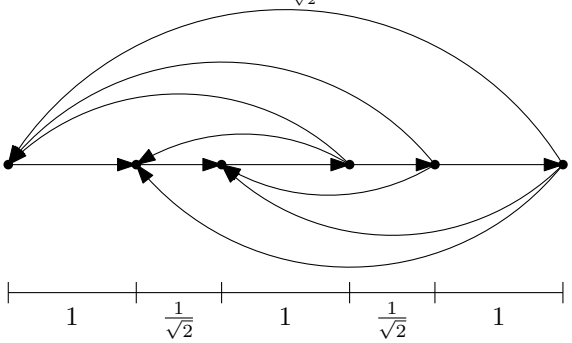
(e) $t = 1 + \frac{\frac{1}{\sqrt{2}}}{1+\frac{1}{\sqrt{2}}} = \sqrt{2}$



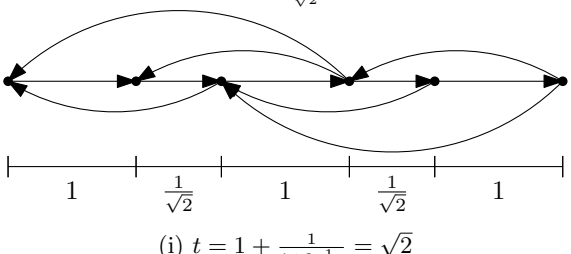
(f) $t = 1 + \frac{1}{1+\frac{1}{\sqrt{2}}} = 3 - \sqrt{2}$



(g) $t = 1 + \frac{1}{1+\frac{1}{\sqrt{2}}} = 3 - \sqrt{2}$

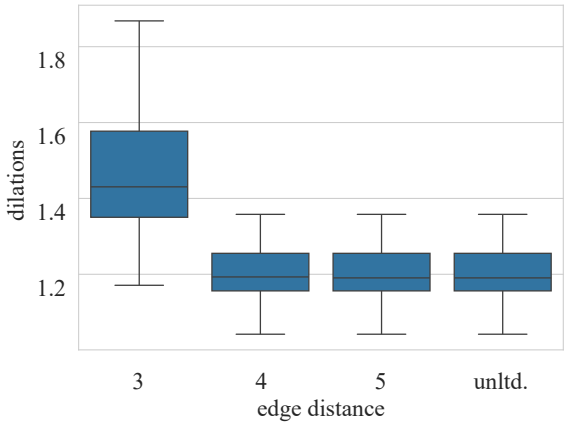


(h) $t = 1 + \frac{1}{1+\frac{1}{\sqrt{2}}} = 3 - \sqrt{2}$

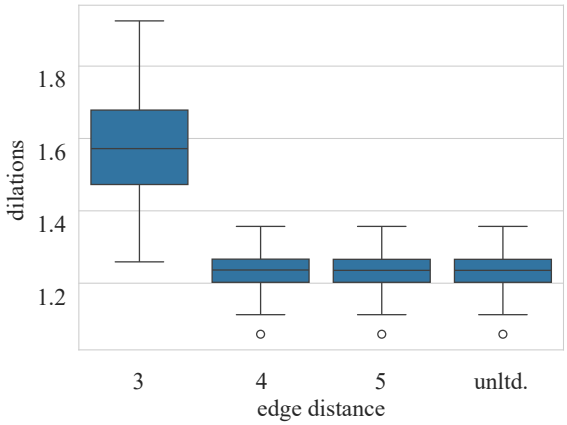


(i) $t = 1 + \frac{1}{1+2\frac{1}{\sqrt{2}}} = \sqrt{2}$

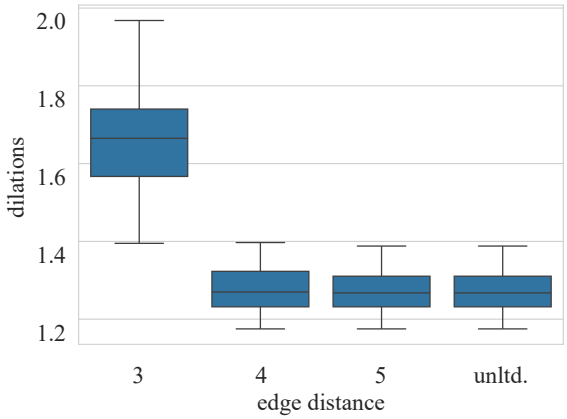
Figure 6: All 2-PPB graphs for P with $p_2 - p_1 = p_4 - p_3 = p_6 - p_5 = 1$ and $p_3 - p_2 = p_5 - p_4 = \frac{1}{\sqrt{2}}$, taking into account the symmetry of P, and their dilation t



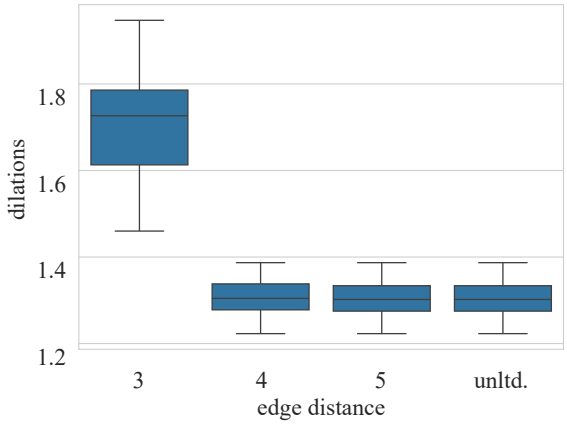
(a) $n = 16$



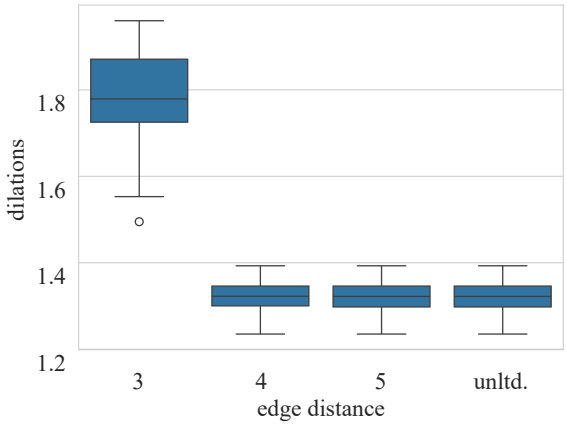
(b) $n = 32$



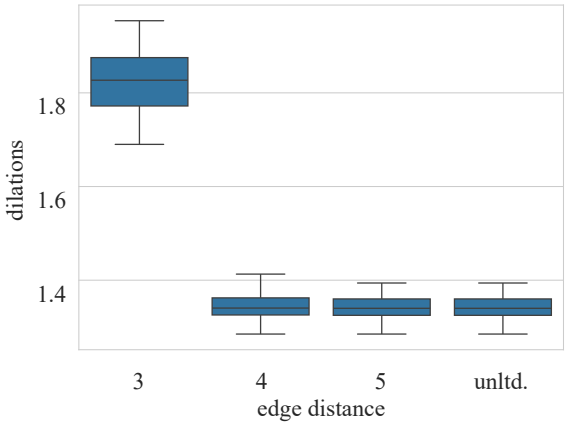
(c) $n = 64$



(d) $n = 128$



(e) $n = 256$



(f) $n = 512$

Figure 7: Boxplots of the oriented dilation of minimum 2-PPB spanners on uniform randomly generated point sets for bounded edge-distance. The x-axis shows the bound on the edge-distance, the y-axis shows the oriented dilation.

Fast Area-Weighted Peeling of Convex Hulls for Outlier Detection*

Vinesh Sridhar† Rolf Svenning‡

Abstract

We present a novel 2D convex hull peeling algorithm for outlier detection, which repeatedly removes the point on the hull that decreases the hull's area the most. To find k outliers among n points, one simply peels k points. The algorithm is an efficient *heuristic* for *exact* methods, which find the k points whose removal together results in the smallest convex hull. Our algorithm runs in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(n)$ space for any choice of k. This is a significant speedup compared to the fastest exact algorithms, which run in $\mathcal{O}\big(n^2 \log n + (n-k)^3\big)$ time using $\mathcal{O}\big(n \log n + (n-k)^3\big)$ space by Eppstein et al. [12, 14], and $\mathcal{O}\Big(n \log n + \binom{4k}{2k}(3k)^k n\Big)$ time by Atanassov et al. [4]. Existing heuristic peeling approaches are not area-based. Instead, an approach by Harsh et al. [17] repeatedly removes the point furthest from the mean using various distance metrics and runs in $\mathcal{O}(n \log n + kn)$ time. Other approaches greedily peel one convex layer at a time [20, 2, 19, 30], which is efficient when using an $\mathcal{O}(n \log n)$ time algorithm by Chazelle [7] to compute the convex layers. However, in many cases this fails to recover outliers. For most values of n and k, our approach is the fastest and first practical choice for finding outliers based on minimizing the area of the convex hull. Our algorithm also generalizes to other objectives such as perimeter.

1 Introduction

When performing data analysis, a critical first step is to identify outliers in the data. This has applications in data exploration, clustering, and statistical analysis [31, 9, 23]. Typical methods of outlier detection such as Grubbs' test [15] are based in statistics and require strong assumptions about the distribution from which the sample is taken. These are known as parametric outlier detection tests. If the sample size is too small or the distribution assumptions are incorrect, parametric tests can produce misleading results. For these reasons, non-parametric complementary approaches based in computation geometry have emerged. Our work follows this line of research and is based on the fundamental notion of a convex hull. For a set of points P, the convex hull is the smallest convex set containing P [10].

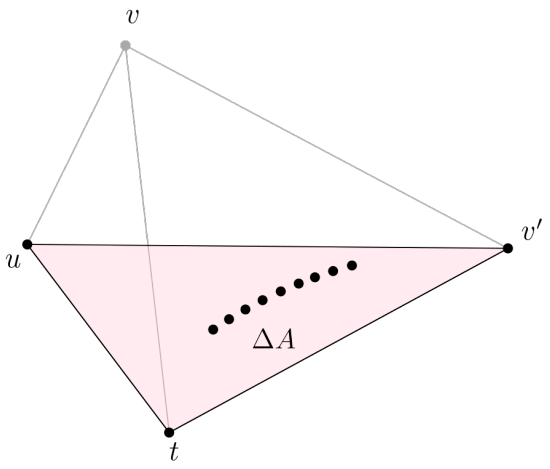


Figure 1: Here point v was peeled from the convex hull and replaced by v'. The previous triangle $\triangle tuv$ for u contained no points. However, when u's triangle becomes $\triangle tuv'$, the set of points ΔA affect the sensitivity $\sigma(u)$ of u. The size of ΔA may be $\Omega(n)$.

There are numerous definitions of outliers [22, 28, 3], but a general theme is that points without many close neighbors are likely to be outliers. As such, these outlying points tend to have a large effect on the shape of the convex hull. Prior work has applied this insight in different ways to identify possible outliers, such as removing points from the convex hull to minimize its diameter [1, 13], its perimeter [11], or its area [14, 12]. Motivated by the last category, we will consider likely outliers to be points whose removal causes the area of the convex hull to shrink the most. We propose a greedy algorithm that repeatedly removes the point $p \in P$ such that the area of P's convex hull decreases the most. We call the amount the area would decrease if point p is removed its *sensitivity* $\sigma(p)$. The removed point is guaranteed to be on the convex hull, and such an algorithm is known as a convex hull *peeling* algorithm [19, 30]. To find k outliers, we peel k points. Our algorithm is conceptually simple, though it relies on the black-box use of a dynamic (or deletion-only) convex hull data structure [18, 6]. We assume that points are in general position. This assumption may be lifted using perturbation methods [25].

*This work is supported in part by Independent Research Fund Denmark grant 9131-00113B and a fellowship from the Department of Computer Science at UC Irvine.

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

‡The Department of Computer Science, Aarhus University, rolfsvenning@cs.au.dk

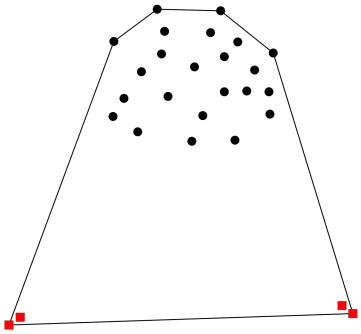


Figure 2: This figure demonstrates the limitations of our heuristic weighted-peeling approach. Clearly, the red squares are outliers, but because there are two squares close-by, the sensitivity of the red squares is minimal. Thus, our algorithm may peel all the valid points before peeling the outlier squares. Note that two k-peels for $k = 2$ would be sufficient to remove all outliers.

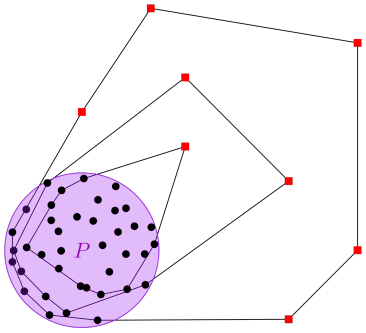


Figure 3: This example shows points drawn uniformly from a target disk P. Clearly, the outliers are the points marked as red squares. It shows the downside of peeling based on depth since many points have to be peeled before reaching the outliers on the second and third layers. In particular, if there are n points drawn uniformly from P, then its convex hull has expected size $\mathcal{O}\big(n^{1/3}\big)$ [16].

The main challenge is maintaining the sensitivities as points are peeled. When peeling a single point v, there may be $\Omega(n)$ new points affecting the sensitivity $\sigma(u)$ for a different point $u \neq v$, as in Figure 1. In that case, naively computing the new sensitivity $\sigma(u)$ would take $\Omega(n)$ time. Nevertheless, we show that our algorithm runs in $\mathcal{O}(n \log n)$ time for any $1 \leq k \leq n$.

2 Related work

The two existing approaches for finding outliers based on the area of the convex hull took a more ideal approach. They considered finding the k points (outliers) whose removal together causes the area of the convex hull to decrease the most. We call this a *k-peel* and note that it always yields an area smaller or equal to that of performing k individual 1-peels. It is not hard to come up with examples where the difference in area between the two approaches is arbitrarily large such as in Figure 2. Still, these examples are quite artificial and require that outliers have at least one other point close by. More importantly, these methods are combinatorial in nature, and much less efficient than our algorithm. The state-of-the-art algorithms for performing a *k-peel* run in $\mathcal{O}\big(n^2 \log n + (n-k)^3\big)$ time and $\mathcal{O}\big(n \log n + (n-k)^3\big)$ space by Eppstein [12, 14] and $\mathcal{O}\Big(n \log n + \binom{4k}{2k}(3k)^k n\Big)$ time by Atanassov et al. [4]. While excellent theoretical results, for most values of $1 \leq k \leq n$ and n, the running time of both of these algorithms is prohibitive for practical purposes. Our contribution is a fast and practical heuristic for these ideal approaches. There are also several results for finding the k points minimizing other objectives such as the minimum diameter, perimeter, or area-enclosing rectangle [13, 29].

Another convex hull peeling algorithm is presented in [17]. Unlike in area-based peeling, they repeatedly remove the point furthest from the mean under various distance metrics. Letting d be the time to compute the distance between two points, their algorithm runs in $\mathcal{O}(n \log n + knd)$ time, which is also significantly slower than our algorithm for most values of k. Since they maintain the mean of the remaining points during the peeling process, each peel takes $\Theta(n)$ time.

Some depth-based outlier detection methods also use convex hulls. They compute a point set's convex layers, which can be defined by iteratively computing $P \setminus CH(P)$ and are computable in $\mathcal{O}(n \log n)$ time [7]. Here, points are deleted from the outermost-layer-in [20, 2, 19, 30]. While efficient, the natural example in Figure 3 is a bad instance for this approach.

3 Results

The main result of our paper is Theorem 1, that there exists an algorithm for efficiently performing area-weighted-peeling.

Theorem 1 *Given n points in 2D, Algorithm 1 performs area-weighted-peeling, repeatedly removing the point from the convex hull which causes its area to decrease the most, in $\mathcal{O}(n \log n)$ time.*

To prove Theorem 1, we derive Theorem 5, which bounds the total number of times points become *active* in any 2D convex hull peeling process to $\mathcal{O}(n)$.

Definition 3.1 (Active Points) *Let (t, u, v) be consecutive points on the first layer in clockwise order. A point p is active for u if, upon deleting u and restoring the first and second layers, p moves to the first layer.*

Intuitively, the active points are the points not on the convex hull that affect the sensitivities. Note that the active points form a subset of the points on the second convex layer. We define $A(u)$ to be the set of active points for point u in a given configuration. Furthermore, all points in $A(u)$ can be found by performing gift-wrapping starting from u's counterclockwise neighbor t while ignoring u. We use this ordering for the points in $A(u)$. In Theorem 7, we show that our algorithm generalizes to other objectives such as *perimeter* where the sensitivity only depends on the points on the first layer and the active points.

4 Machinery

In this section, we describe some of the existing techniques we use. To efficiently calculate how much the hull shrinks when a point is peeled, we perform tangent queries from the neighbours of the peeled point to the second convex layer. The tangents from a point q to a convex polygon L can be found in $\mathcal{O}(\log n)$ time both with [27] and without [21] a line separating q and L. In our application, such a separating line is always available, and either approach can be used. Tangent queries require that L is represented as an array or a balanced binary search tree of its vertices ordered (cyclically) as they appear on the perimeter of L. To allow efficient updates to L we use a binary tree representation that is leaf-linked such that given a pointer to a vertex its successor/predecessor can be found in $\mathcal{O}(1)$ time.

The convex layers of n points can be computed in $\mathcal{O}(n \log n)$ time using an algorithm by Chazelle [7]. Given l convex layers, after a single peel they can be restored in $\mathcal{O}(l \log n)$ time (Lemma 3.3 [24]). However, for our purposes we only need the 2 outermost layers for area calculations. As such, we explicitly maintain the two outermost layers L^1 and L^2, and we store all remaining points $P \backslash \{L^1 \cup L^2\}$ in a *center* convex hull. To restore L^1 we use tangent queries on L^2 as in [24]. To restore L^2 we use extreme point queries on the center convex hull which we maintain using a semi-dynamic [18] or fully-dynamic [6] convex hull data structures supporting extreme point queries in worst case $\mathcal{O}(\log n)$ time and updates in amortized $\mathcal{O}(\log n)$ time.

5 Area-Weighted-Peeling Algorithm

In this section, we describe Algorithm 1 in detail and show that its running time is $\mathcal{O}(n \log n)$.

At a high level, we want to repeatedly identify and remove the point which causes the area of the convex hull to decrease the most. Such an iteration is a *peel*, and we call the amount the area would decrease if point u was peeled the sensitivity $\sigma(u)$ of u. To efficiently find the point to peel, we maintain a priority queue Q on the sensitivities of hull points. Only points on the convex hull may have positive sensitivity, and in lines 2-6 we compute the initial sensitives of the points on the convex hull and store them in Q. For a hull point u, to compute its sensitivity $\sigma(u)$ we find its active points $A(u)$. Note they must be on the second convex layer, and if u's neighbors are t and v, then the points $A(u)$ are in the triangle $\triangle tuv$. In line 1 we compute the two outer convex hull layers represented as balanced binary trees. That allows us to compute $A(u)$ using tangent queries on the inner layer from t and v. Then $\sigma(u)$ can be found by computing the area of the polygon $\pentagon(t \circ v \circ A(u))$.

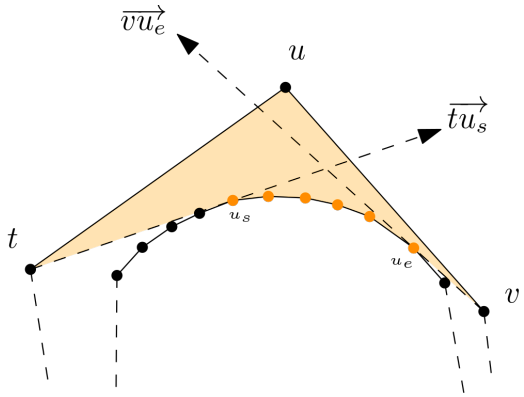


Figure 4: Using u's neighbors, we can perform two tangent queries on L^2 to recover the first and last active point of u, labeled u_s and u_e respectively, in $\mathcal{O}(\log n)$ time. Because we represent L^2 as a leaf-linked tree, we can walk along L^2 to recover all points of $A(u)$. The shaded part of the figure represents $\sigma(u)$.

As points are peeled (lines 8-17) layers L^1 and L^2 must be restored. To restore L^1 when point u is peeled (line 9) we perform tangent queries on L^2 as in [24] to find u's active points $A(u)$ (line 10) and move $A(u)$ from L^2 to L^1. See Figure 4 for an example of tangent queries from L^1 to L^2.

To restore the broken part of L^2, we perform extreme point queries on the remaining points efficiently using a dynamic convex hull data structure D_{CH} (line 7) as in [18] or [6]. As described in Lemma 6, $A(u)$ is always contiguous on L^2. Therefore, removing $A(u)$ from L^2 requires us to restore it between two "endpoints" a and b. The first extreme point query uses line $\overline{ab}$ in the direction of u. If a point z from D_{CH} is found then at least two more queries are performed with lines $\overline{za}$ and $\overline{zb}$. In general, if k points are found then the number of queries is $2k + 1$. The k points are deleted from D_{CH}. This all happens on line 11.

Next, we compute the sensitivities of the new points on the hull (line 14) and insert them into the priority queue. Finally, we update the sensitivities of u's neighbors t and v (line 17), which, by Lemma 2(4), are the only two points already in Q whose sensitivity changes.

Algorithm 1: Weighted peeling

Input: A set of n points P in 2D

1 $L^1, L^2 \longleftarrow$ the first two convex layers of P
2 $Q \longleftarrow$ empty max priority queue
3 **for** $i = 1$ **to** $\left|L^1\right|$ **do**
4 $u \longleftarrow L^1_i$
5 Compute sensitivity $\sigma(u)$ for u
6 $Q.insert(u, \sigma(u))$
7 $D_{CH} \longleftarrow$ a dynamic convex hull data structure on $P \setminus \left\{L^1 \cup L^2\right\}$
8 **for** $i = 1$ **to** n **do**
9 $u \longleftarrow Q.extractMax$
10 $A(u) \longleftarrow u$'s active points
11 Delete u from L^1 and update L^1, L^2 and D_{CH}
12 **for** $i = 1$ **to** $|A(u)|$ **do**
13 $\bar{u} \longleftarrow A(u)_i$
14 Compute sensitivity $\sigma(\bar{u})$ for $\bar{u}$
15 $Q.insert(\bar{u}, \sigma(\bar{u}))$
16 $t, v \longleftarrow$ neighbors of u in L^1
17 Update $Q[t]$ and $Q[v]$

5.1 Analysis

The hardest part of the analysis is showing that the overall time spent on lines 14 and 17 is $\mathcal{O}(n \log n)$. We first show that, excluding the time spent on these lines, the running time of Algorithm 1 is $\mathcal{O}(n \log n)$. In line 1 we compute the first and second convex layers in $\mathcal{O}(n \log n)$ time by running any optimal convex hull algorithm twice. In lines 2 to 6, we compute the initial sensitivities by finding the points active for each $u \in L^1$. As described above, we can do this by applying two tangent queries, allowing us to recover the first and last extreme point for u. We can walk along L^2 between them to recover $A(u)$. Once $A(u)$ is found for each u, we find $\sigma(u)$ by computing the area of the polygon $\pentagon(t \circ u \circ v \circ A(u))$, where t and v are u's neighbors. By Lemma 2(1), in this initial configuration each point on L^2 is active in at most three triangles. Thus, we make in total $\mathcal{O}\left(\left|L^2\right|\right) = \mathcal{O}(n)$ tangent queries, each of which costs $\mathcal{O}(\log n)$ time. Since the area of a simple polygon can be computed in linear time [26], all the area computations take $\sum_{u \in L^1} \Theta(1 + |A(u)|) = \mathcal{O}\left(\left|L^1\right| + \left|L^2\right|\right) = \mathcal{O}(n)$ time. Therefore, the overall time to initialize the priority queue is $\mathcal{O}(n \log n)$.

Initializing D_{CH} in line 7 takes $\mathcal{O}(n \log n)$ time [18]. In line 10, we can perform tangent queries on L^2 from t and v to find the first and last active points of u. In line 11, it will take no more than $\mathcal{O}(n)$ tangent queries to restore L^1 and L^2 throughout the algorithm by charging the queries to the points moved from the center convex hull to L^2 or from L^2 to L^1. Using an efficient dynamic convex hull data structure, it takes $\mathcal{O}(\log n)$ amortized time to delete a point and thus $\mathcal{O}(n \log n)$ time overall [18, 6]. We add points to the priority queue n times, delete points from the priority queue n times, and perform $\mathcal{O}(1)$ priority queue update operations for each iteration of the outer loop on line 8. Excluding lines 14 and 17 this establishes the overall $\mathcal{O}(n \log n)$ running time.

To bound the total time spent on line 14 to $\mathcal{O}(n \log n)$, we prove Theorem 5, bounding the total number of times points becomes active to $\mathcal{O}(n)$. Computing $\sigma(\bar{u})$ in line 14 requires us to find $A(\bar{u})$, where $\bar{u}$ is a new point added to the first layer. From the theorem, it takes $\mathcal{O}(n \log n)$ time to compute $A(\bar{u})$ for every $\bar{u}$. In addition, because it takes $\Theta(1 + |A(\bar{u})|)$ to compute $\sigma(\bar{u})$ from $A(\bar{u})$, overall it takes $\mathcal{O}(n)$ time to compute $\sigma(\bar{u})$ for every $\bar{u}$.

To bound the total time spent on line 17 on updating the sensitivities of u's neighbors to $\mathcal{O}(n \log n)$, we prove Lemma 6. Together with Theorem 5, it implies the desired result.

6 Geometric properties of peeling

In this section, we develop an amortized analysis of peeling to show that lines 14 and 17 can be computed efficiently. We ultimately aim to show that the number of times that any point becomes active for any triangle is $\mathcal{O}(n)$, bounding the amount of work done to initialize new triangles to $\mathcal{O}(n \log n)$. Then we show that the amount of work done to update the sensitivities of neighbor points is proportional to the number of new active points for them and an additive $\mathcal{O}(\log n)$ term. Thus, updating the sensitivities over all n iterations takes $\mathcal{O}(n \log n)$.

6.1 Preliminaries

When considering outer hull points, we use the notation $\triangle tuv$ for the triangle formed by u, its counterclockwise neighbor t, and its clockwise neighbor v. For a set of ordered vertices V we let $\pentagon(V)$ be the polygon formed by the points in the (cyclical) order. We say $p \in \pentagon(V)$ if p is strictly inside the polygon.

The following Lemma 2 combines a number of simple but useful propositions.

Lemma 2 *For a set of points P, the following propositions are true:*

1. *Any point $p \in P$ is active for at most three points on the first layer.*
2. *Let $\triangle tuv$ be a triangle for consecutive vertices (t, u, v) on the first layer and let $p \neq q$ be points $p \in \triangle tuv$ and $q \in \triangle tpv$. Then $q \notin A(u)$.*

3. *Let p be a point on any layer k. After deleting any point $q \neq p$ and reconstructing the convex layers, p is on layer $k-1$ or k.*

4. *Let (t, u, v) be consecutive vertices on the first layer L^1. Then if u is deleted, among the vertices in L^1, only the sensitivities of vertices t and v change.*

5. *For adjacent points (u, v) on the hull, $|A(u) \cap A(v)| \leq 1$.*

Proof. See Section 7.2 in the appendix. □

6.2 Bounding the active points

We will show that once a point is active for a hull point, it remains active for that hull point until the point is moved to the first layer. This implies a much stronger result by Lemma 2(1): over the entire course of the algorithm, a point becomes active for at most three other points. To do so, we first show that for each peel the active points $A(u)$ remain in u's triangle (Lemma 3) and second that the points in $A(u)$ remain active (Lemma 4).

Lemma 3 *Given a set of points P, for all adjacent hull points (u, v) and for all points $p \in A(u) \setminus A(v)$, if v is deleted then p still remains within u's triangle.*

Proof. Let t be u's other neighbor, and w.l.o.g. let the clockwise order on the hull be (t, u, v). Then if v' is u's new neighbor after deleting v, the clockwise order on the new hull will be (t, u, v'). Because p is active for u before v is deleted, $p \in \triangle tuv$.

First, we consider the case where $v' \notin \triangle tuv$. We want to show that $p \in \triangle tuv'$. Equivalently, that p is in the intersection of the three half-planes $\overrightarrow{tu}$, $\overrightarrow{uv'}$, and $\overrightarrow{tv'}$. Clearly, p must satisfy the half-planes $\overrightarrow{tu}$ and $\overrightarrow{uv'}$ as these coincide with hull edges. In addition, since $v' \notin \triangle tuv$, the half-plane for $\overrightarrow{tv}$ is a subset of the half-plane for $\overrightarrow{tv'}$. Because $p \in \triangle tuv$, p satisfies $\overrightarrow{tv}$. Therefore, p must satisfy $\overrightarrow{tv'}$.

Now we consider the case where $v' \in \triangle tuv$. Assume that $p \notin \triangle tuv'$. Then because we know that $p \in \triangle tuv$, either $p \in \triangle tv'v$ or $p \in \triangle uv'v$. If $p \in \triangle tv'v$, by Lemma 2(2), p could not have been active for u prior to deleting v. If $p \in \triangle uv'v$, p is now outside of the convex hull. Either way, this is a contradiction.

□

The following Lemma 4 shows that if p is in $A(u)$, it remains in $A(u)$ until moved to the first layer, after which it never becomes active again. It also shows that the active points $A(u)$ only change by adding or deleting points from either end, and thus can easily be found.

Lemma 4 *Given a set of points P, for all hull points u and v and for all points $p \in A(u) \setminus A(v)$, upon deleting v, p is in $A(u)'$, u's new set of active points.*

Proof.

Case 1 (u is not adjacent to v)

If u is not adjacent to v, there are no changes to $\triangle u$ upon deleting v, and thus, $A(u) = A(u)'$.

For the following cases, assume that u was adjacent to v. Then by Lemma 3, p is still in the triangle defined by u even after deleting v. Also, w.l.o.g. let (u, v) be the clockwise ordering of the points, and let v' be u's new neighbor.

Case 2 ($v' \in A(u)$)

By Lemma 2(5), $A(u) \cap A(v) = v'$. By Lemma 3, all points $A(u) \setminus \{v'\}$ are in $\triangle tuv'$. Because the second layer is a convex hull, each consecutive pair of points (a, b) in $t \circ A(u)$ define a half-plane $\overrightarrow{ab}$ with only points from the first layer to the left of each half-plane. This is still the case after deleting v by Lemma 3. Since the only new points on the first layer are $A(v)$ then all points in $A(u) \setminus \{v'\}$ remain on the second layer. Thus, the gift-wrapping starting from t wraps around all points in $A(u) \setminus \{v'\}$. Gift wrapping can hit no new points because, if that were true, there must be some point on the second layer to the left of one of the half-planes in described above. Thus, $A(u)' = A(u) \setminus \{v'\}$.

Case 3 ($v' \notin A(u)$)

Let u_e be the last point $A(u)$. Similar to the previous case, the gift-wrapping certifies all points in $A(u)$. Again, wrapping will not hit new active points before wrapping around u_e because that would imply the points hit were to the left of the half-planes described previously. When wrapping continues around u_e, several new active points may appear, until the wrapping terminates at v'. Thus, $A(u) \subseteq A(u)'$. □

Theorem 5 *For any 2D convex hull peeling process on n points the total number of times any point becomes active in any triangle is at most $3n$.*

Proof. This follows directly from the results of Lemma 2(1) and Lemma 4. □

6.3 Updating sensitivities

Next, we show that the total time to update the sensitivities in line 17 when peeling all n points takes $\mathcal{O}(\Delta + n \log n)$ time. Here Δ is the the number of times any point becomes active for any triangle. Theorem 5 proves that $\Delta = \mathcal{O}(n)$. The following lemma shows that the sensitivity of a point u can be updated in time proportional to the increase to $|A(u)|$ and an additive $\mathcal{O}(\log n)$ term. Figure 5 shows an example of how the sensitivity of a point changes when its neighbor is peeled.

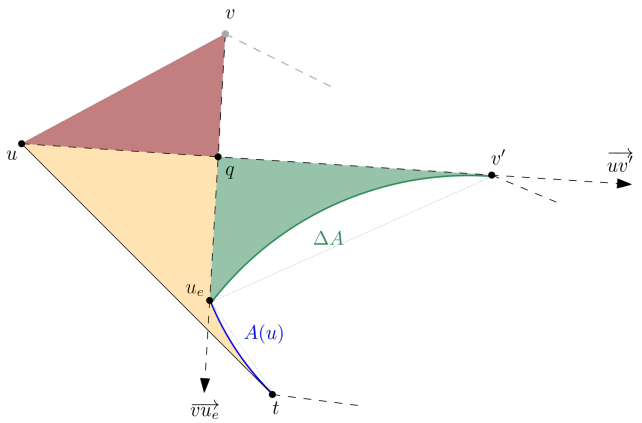


Figure 5: This figure shows how the sensitivity $\sigma(u)$ changes when point v is peeled. The point q is the intersection of the tangent from v to u_e and the tangent from u to v', where u_e is the last active point in $A(u)$ and v' is the first active point in $A(v)$. After the peel, v' replaces v as u's neighbor, and the points ΔA are newly active for u. The sensitivity $\sigma(u)$ before peeling v was equal to the area of $\text{⬠}(t \circ u \circ v \circ A(u))$. After peeling v, the sensitivity $\sigma(u)$ equals the area of $\text{⬠}(t \circ u \circ v' \circ \Delta A \circ A(u))$. Note how this can be computed in $\mathcal{O}(|\Delta A|)$ time from $\sigma(u)$ before the peel of v by subtracting the red area of $\triangle uvq$ and adding the green area of $\text{⬠}(u_e \circ q \circ v' \circ \Delta A)$.

Lemma 6 *Let (u, v) be points on the first layer. Consider a peel of v where δ_u new points become active points for u. Then the updated sensitivity $\sigma(u)$ can be computed it $\Theta(\delta_u + \log n)$ time, excluding the time to restore the second and first layer.*

Proof. The sensitivity $\sigma(u)$ is equal to the area of the polygon $U = \text{⬠}(t \circ u \circ v \circ A(u))$. By the *shoelace formula*, the area of U can be computed as the sum $S(U)$ of certain simple terms for each of its edges [5, 8]. We consider how U, and thus $S(U)$, changes when v is peeled. Inspecting the proof of Lemma 4, we see that at most two vertices are removed from U and at most $1 + \delta_u$ vertices are added to U. Furthermore, all the new vertices are located contiguously on the restored second layer and can be found in $\mathcal{O}(\delta_u + \log n)$ time using a tangent query from u's new neighbor which replaces v. To update $\sigma(u) = S(U)$, we simply add and subtract the appropriate $\mathcal{O}(\delta_u)$ terms depending on the removed and added edges.

□

7 Generalization and open problems

Theorem 7 shows that Algorithm 1 generalizes straightforwardly to other objectives such as peeling the point that causes the perimeter of the convex hull to decrease the most each iteration.

Theorem 7 *Let u be a point and O an objective where $\sigma_O(u)$ is the sensitivity of u under O. Consider the following three conditions:*

C1: *If $u \notin L^1$, then $\sigma_O(u) = 0$.*

C2: *If $u \in L^1$, then $\sigma_O(u) > 0$, and $\sigma_O(u)$ depends only on u, u's neighbors and its active points $A(u)$.*

C3: *If a single point p is added or removed from $A(u)$, then provided $\sigma_O(u)$ and the neighbors a_i and a_j of p in $A(u)$, the new sensitivity $\sigma_O(u)'$ can be computed in $\mathcal{O}(\log n)$ time.*

If O satisfies the above conditions, then Algorithm 1 runs in $\mathcal{O}(n \log n)$ time for objective O.

Proof. By conditions **C1** and **C2**, it is always a point u on the first layer that is peeled. Furthermore, when u is peeled only the sensitivities of the new points on the first layer and the neighbors of u must be updated since they are the only points for which their active points or neighbors change. Thus, Algorithm 1 can be used for objective O. Now we will show that the runtime of Algorithm 1 remains $\mathcal{O}(n \log n)$.

First, observe that all parts unrelated to computing sensitivities behave the same and still take $\mathcal{O}(n \log n)$ time. By condition **C3**, for a point u on the first layer, its sensitivity $\sigma_O(u)$ only depends on its neighbors and active points $A(u)$. As described in the proof of Lemma 6, when the set of points that affect $\sigma_O(u)$ changes, these points are readily available. The total number of neighbor changes is $\mathcal{O}(n)$ since, in each iteration, only the neighbors of the points adjacent to the peeled point change. The total number of changes to active points is $\mathcal{O}(n)$ by Theorem 5. If there are multiple changes to the active points in one iteration, such as when deleting one of u's neighbors, we perform one change at a time and, by condition **C3**, the total time to update sensitivities is $\mathcal{O}(n \log n)$. □

For concrete examples, we show how the three objectives *area* (O_A), *perimeter* (O_P), and *number of active points* (O_N) fit into this framework.

Let $f(\sigma(u), a_i, p, a_j) = \sigma(u) - d(a_i, a_j) + d(a_i, p) + d(p, a_j)$ be a function for computing the sensitivity $\sigma(u)$ when p is added to $A(u)$ between a_i and a_j (the functions where a point is removed from $A(u)$ or a neighbor of u changes are similar). For f to match each of the objectives it is sufficient to implement $d(\cdot, \cdot)$ as follows for points $a, b \in \mathbf{R}^2$:

O_A: $d(a, b) = \frac{1}{2}(a_2 b_1 - a_1 b_2)$

O_P: $d(a, b) = \sqrt{(b_2 - a_2)^2 + (b_1 - a_1)^2}$

O_N: $d(a, b) = 1$

The case with O_A is based on the shoelace formula. Additionally, for O_N to satisfy condition **C2**, we add 1 when computing the sensitivity of $u \in L^1$ to ensure that $\sigma(u) > 0$ even if $|A(u)| = 0$. For the three objectives, f takes $\mathcal{O}(1)$ time to compute satisfying the $\mathcal{O}(\log n)$ time requirement from condition **C3**.

7.1 Open problems

The first open problem is extending the result to $\mathbf{R}^3$ or higher. Directly applying our approach requires a dynamic 3D convex hull data structure, and Theorem 5 has to be extended to 3D. Second, is it possible to improve the quality of peeling by performing z-peels, even for $z = 2$ in $o(n)$ time? Third, is there an efficient approximation algorithm for k-peeling?

Acknowledgement

We thank Asger Svenning for the initial discussions that inspired us to consider this problem.

References

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Fining k points with minimum spanning trees and related problems. In *Proceedings of the fifth annual symposium on Computational geometry*, pages 283–291, 1989.

[2] G. Aloupis. Geometric measures of data depth. *DIMACS series in discrete mathematics and theoretical computer science*, 72:147, 2006.

[3] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *European conference on principles of data mining and knowledge discovery*, pages 15–27. Springer, 2002.

[4] R. Atanassov, P. Bose, M. Couture, A. Maheshwari, P. Morin, M. Paquette, M. Smid, and S. Wuhrer. Algorithms for optimal outlier removal. *Journal of Discrete Algorithms*, 7(2):239–248, 2009. Selected papers from the 2nd Algorithms and Complexity in Durham Workshop ACiD 2006.

[5] R. Boland and J. Urrutia. Polygon area problems. In *Proc. of the 12th Canadian Conf. on Computational Geometry*, Fredericton, NB, Canada, 2000.

[6] G. Brodal and R. Jacob. Dynamic planar convex hull. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 617–626, 2002.

[7] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 1985.

[8] F. Contreras. *Cutting polygons and a problem on illumination of stages.* University of Ottawa (Canada), 1998.

[9] R. N. Dave. Characterization and detection of noise in clustering. *Pattern Recognition Letters*, 12(11):657–664, 1991.

[10] M. De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

[11] D. P. Dobkin, R. Drysdale, and L. J. Guibas. Finding smallest polygons. *Computational Geometry*, 1:181–214, 1983.

[12] D. Eppstein. New algorithms for minimum area k-gons. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, page 83–88, USA, 1992. Society for Industrial and Applied Mathematics.

[13] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11:321–350, 1994.

[14] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete & Computational Geometry*, 7:45–58, 1992.

[15] F. E. Grubbs. *Sample criteria for testing outlying observations.* University of Michigan, 1949.

[16] S. Har-Peled. On the expected complexity of random convex hulls. *arXiv preprint arXiv:1111.5340*, 2011.

[17] A. Harsh, J. E., and P. Wei. Onion-peeling outlier detection in 2-d data sets. *International Journal of Computer Applications*, 139:26–31, 04 2016.

[18] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Numerical Mathematics*, 32:249–267, 1992.

[19] P. J. Huber. The 1972 wald lecture robust statistics: A review. *The Annals of Mathematical Statistics*, 43(4):1041–1067, 1972.

[20] J. Hugg, E. Rafalin, K. Seyboth, and D. Souvaine. An experimental study of old and new depth measures. In *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 51–64. SIAM, 2006.

[21] D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In S. G. Akl, F. Dehne, J.-R. Sack, and N. Santoro, editors, *Algorithms and Data Structures*, pages 183–193, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[22] E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. In *Vldb*, volume 99, pages 211–222, 1999.

[23] S. K. Kwak and J. H. Kim. Statistical data preparation: management of missing values and outliers. *Korean journal of anesthesiology*, 70(4):407, 2017.

[24] M. Löffler and W. Mulzer. Unions of onions: preprocessing imprecise points for fast onion decomposition. *Journal of Computational Geometry*, 5(1), 2014.

[25] K. Mehlhorn, R. Osbild, and M. Sagraloff. Reliable and efficient computational geometry via controlled perturbation. In *International Colloquium on Automata, Languages, and Programming*, pages 299–310. Springer, 2006.

[26] A. Meister. *Generalia de genesi figurarum planarum et inde pendentibus earum affectionibus.* 1769.

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

[28] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438, 2000.

[29] M. Segal and K. Kedem. Enclosing k points in the smallest axis parallel rectangle. *Information Processing Letters*, 65(2):95–99, 1998.

[30] M. I. Shamos. *Problems in computational geometry.* 1975.

[31] A. F. Zuur, E. N. Ieno, and C. S. Elphick. A protocol for data exploration to avoid common statistical problems. *Methods in ecology and evolution*, 1(1):3–14, 2010.

Appendix

7.2 Proof of Lemma 2

Lemma 2(1) *Fix a point set P. Any point $p \in P$ is active in at most three triangles.*

Proof. First, note that a point can only be active for a hull point u if it is located inside $\triangle u$, so it is sufficient to show that any p is strictly inside at most three triangles. In addition, one can prove this by showing that $\triangle u$ only intersects with its neighbors' triangles $\triangle t$ and $\triangle v$.

Consider some $\triangle z$, such that z is not a neighbor of u. That is, u is not one of the vertices of $\triangle z$. If $\triangle z$ intersects with $\triangle u$, then either a vertex of $\triangle z$ is inside $\triangle u$ or the convex hull is a self-intersecting polygon, both violating convexity.

□

Lemma 2(2) *Let $\triangle tuv$ be a triangle for consecutive vertices (t, u, v) on the first layer and let $p \neq q$ be points $p \in \triangle tuv$ and $q \in \triangle tpv$. Then $q \notin A(u)$.*

Proof. By definition, $p \in \hexagon(t \circ v \circ A(u))$ or $p \in A(u)$. Either way, $q \in \triangle tpv$ implies that $q \in \hexagon(t \circ v \circ A(u))$, so $q \notin A(u)$. □

Lemma 2(3) *Let p be a point on any convex layer k. After deleting any point $q \neq p$ and reconstructing the convex layers, p is on layer $k-1$ or k.*

Proof. First we show that p never moves inward to layer $k' > k$. Consider the outermost layer L^1. By a property of convex hulls, every point v inside the convex hull is a convex combination of the hull points whereas any point $u \in L^1$ is not a convex combination of $L^1 - \{u\}$. If deleting q causes $p \in L^1$ to descend to a layer inside L^1, that implies that p is a convex combination of some subset of $P - \{q, p\}$. This contradicts the fact that p is not a convex combination of $L^1 - \{p\}$ and by extension is not a convex combination of $P - \{p\}$. Because of the recursive definition of convex layers, the proof for subsequent layers is symmetric.

Now we will show that p never moves up more than one layer at a time. This is clearly true for L^1 and L^2 because only one point is completely removed from the structure at at time (i.e. shifts to layer 0). For layers $k \geq 3$, consider a point p on layer k that moves to layer $k' \leq k-2$. Let $L^{*k'}$ be the set of points on layer k' after deleting q. Let L^{k-1} be the set of points on layer $k-1$ before deleting q.

Because $p \in L^{*k'}$, no convex combination of the points in $L^{*k'} - \{p\}$ equals p by convexity. By the inductive hypothesis, all points on L^{k-1} are convex combinations of $L^{*k'}$ because upon deleting q no point on L^{k-1} advances above layer k'. Furthermore, they are all convex combinations of $L^{*k'} - \{p\}$ as p itself is a convex combination of L^{k-1}. But if p is not a convex combination of $L^{*k'} - \{p\}$, and all the points on layer $k-1$ are convex combinations of $L^{*k'} - \{p\}$, then prior to deleting q, p was above layer $k-1$, which is a contradiction. □

Lemma 2(4) *Let (t, u, v) be consecutive vertices on the first layer L^1. Then if u is deleted, among the vertices in L^1, only the sensitivities of vertices t and v change.*

Proof. Consider a vertex z not adjacent to u. By the same arguments as in the proof of Lemma 2(1), the vertices defining $\triangle z$ do not change upon deleting u because it does not intersect $\triangle u$. In addition, because their triangles do not intersect, $|A(u) \cap A(z)| = 0$. Therefore, no points are removed from $A(z)$ upon deleting u.

Lastly, we will show that no points are added to $A(z)$ upon deleting u. Assume that there is some point p added to $A(z)$ when we delete u. But if p satisfies the conditions of being active for z and $\triangle z$ did not change upon deleting u, it should have been active for z before u was deleted, which is a contradiction.

Because $\triangle z$ and $A(z)$ do not change upon deleting u, it must be that $\sigma(z)$ remains the same. □

Lemma 2(5) *For adjacent points (u, v) on the hull, $|A(u) \cap A(v)| \leq 1$.*

Proof. We assume the contrary. Let $p \neq p'$ be two points such that $p, p' \in A(u) \cap A(v)$. By the definition of *active* and Lemma 2(3), p and p' must be on the second layer. W.l.o.g. let (u, v) be the clockwise ordering of the points on the first layer. In addition, let t be u's counterclockwise neighbor.

Say that p is the first point in $A(v)$. Then we have the tangent line $\overrightarrow{up}$ that defines p. By definition of tangent lines, no point on the second layer can be to the left of $\overrightarrow{up}$. But for p' to be active for v, then p' must be to the left of $\overrightarrow{pv}$. The only way to satisfy both half-planes is for p' to be placed such that $p \in \triangle tp'v$, in which case by Lemma 2(2) p cannot be in $A(u)$, which is a contradiction.

□

Geometric Localization of Homology Cycles

Amritendu Dhar* Vijay Natarajan† Abhishek Rathod‡

Abstract

Computing an optimal cycle in a given homology class, also referred to as the homology localization problem, is known to be an NP-hard problem in general. Furthermore, there is currently no known optimality criterion that localizes classes geometrically and admits a stability property under the setting of persistent homology. We present a geometric optimization of the cycles that is computable in polynomial time and is stable in an approximate sense. Tailoring our search criterion to different settings, we obtain various optimization problems like optimal homologous cycle, minimum homology basis, and minimum persistent homology basis. In practice, the (trivial) exact algorithm is computationally expensive despite having a worst case polynomial runtime. Therefore, we design approximation algorithms for the above problems and study their performance experimentally. These algorithms have reasonable runtimes for moderate sized datasets and the cycles computed by these algorithms are consistently of high quality as demonstrated via experiments on multiple datasets.

1 Introduction

Homology groups and their persistent version called *persistent homology* play a central role in topological data analysis (TDA), a thriving research field of equal interest to computer scientists, mathematicians and data scientists [17, 18]. The ranks for homology groups and the barcodes for persistent homology groups have been extensively studied both from algorithmic and mathematical perspectives. With the growth of TDA in applications, there is an increasing need for computing homology cycles that localize given homology classes or constitute a basis for the homology group. Often applications require these cycles to be tightest possible or geometry-aware in some sense rather than being completely oblivious of the embedding space. This demand has led to studying homologous or basis cycles under various optimization criteria. A number of optimization results in this direction have now appeared in the literature both in persistent and non-persistent settings [4, 5, 7, 8, 10–13, 16, 27].

The quality of the optimal cycles depends on the choice of a weight function. For instance, one may choose a weight for each p-cycle ζ to be the sum of non-negative weights assigned to each p-simplex in ζ. Optimizing this measure over a class of a given cycle ζ localizes the class $[\zeta]$ in the sense that it selects a cycle in the class with the least weight. Unfortunately, this problem is known to be NP-hard in general [7, 10] except for some special cases [13, 15]. Polynomial time algorithms are known for certain optimization criteria [9, 15] or in lower dimensions [4, 7, 16, 20].

Outline and Contributions. Precisely, we achieve the following. Given a simplicial complex K with the vertices in a point set $P \subset \mathbb{R}^d$ and linearly embedded simplices, we define the weight of a cycle ζ as the radius of the smallest $(d-1)$-sphere that encloses ζ. This measure, in some sense, captures the locality of ζ with respect to its geometry. In Section 4, we study how homology localization serves as an archetype application. Then, we solve other versions of the optimal cycle problem including *minimum homology basis* in Section 5 and *minimum persistent homology basis* in Section 6. For the persistent version, in Appendix B, we show optimal persistent homology bases are stable in an approximate sense. For previous results on optimal persistent cycles [11,15] such stability is not known. The approximation algorithms described in this paper have been implemented. In Section 7, we report experimental results for the approximate algorithms. In our experiments, we found that even the approximate algorithms return cycles of consistently high quality confirming the value of ℓ_2–radius as an optimization criterion. We further compare experimental results on persistent homology with that of PersLoop [28], which is a state of the art software for computing optimal persistent 1-cycles. We visually infer that our cycles are "tighter" than those of PersLoop on multiple datasets of practical importance.

Related work. A criterion related to ours was considered by Chen and Freedman [9] who proposed to compute a minimum homology basis while optimizing

*Department of Computer Science and Automation, Indian Institute of Science, Bangalore, amritendud@iisc.ac.in

†Department of Computer Science and Automation, Indian Institute of Science, Bangalore and Zuse Institute, Berlin, vijayn@iisc.ac.in

‡Department of Computer Science, Ben Gurion University of the Negev, arathod@post.bgu.ac.il

the shortest path radius of the geodesic balls containing the basis cycles. With an embedding in the Euclidean space, the ℓ_2–radius of the geometric balls capture locality more concisely than the shortest path radius. Yet another measures of optimality for cycles that is tractable, namely lexicographically optimality [11], suffers from the drawback that it requires a parameter: a total order on simplices. In applications, it is sometimes desirable that the optimal cycles be stable with regard to the change in the input data [2]. An optimization criterion that is geometry-aware, polynomial time computable, and results in some kind of stability is *volume optimal cycles* by Obayashi [25, 26]. However, unlike our measure, the approach described in [25] works only for computing representatives of finite bars. In another related work, Li et al. [24] obtain minimal representatives using linear programming for a variety of optimization criteria with impressive runtimes. However, their software does not work for arbitrary filtrations yet [22]. In summary, our key contribution in this work is that we introduce a natural measure of optimality of cycles that has good theoretical properties and is well-behaved in practice.

2 Background and preliminaries

In this section, we recall some preliminaries on persistent homology. For the rest of this section, we work only with simplexwise filtrations: That is, we have a filtration $\mathcal{F}$ on $\mathbb{R}$ where the complexes change only at finite set of values $a_1 < a_2 < \ldots < a_n$ and every change involves addition of a unique simplex σ_{a_i} for $i \in [n]$.

$$\mathcal{F} : \emptyset = \mathsf{K}_{a_0} \overset{\sigma_{a_0}}{\hookrightarrow} \mathsf{K}_{a_1} \overset{\sigma_{a_1}}{\hookrightarrow} \mathsf{K}_{a_2} \overset{\sigma_{a_2}}{\hookrightarrow} \ldots \overset{\sigma_{a_{n-1}}}{\hookrightarrow} \mathsf{K}_n = \mathsf{K}$$

Using p-th homology groups of the complexes over the field $\mathbb{Z}_2$, we get a sequence of vector spaces connected by inclusion-induced linear maps:

$$H_p\mathcal{F} : H_p(\mathsf{K}_{a_0}) \to H_p(\mathsf{K}_{a_1}) \to H_p(\mathsf{K}_{a_2}) \to \ldots$$

The sequence $H_p\mathcal{F}$ with the linear maps is called a *persistence module*. There is a special persistence module called the *interval module* $\mathbb{I}^{[b,d)}$ associated to the interval $[b, d)$. Denoting the vector space indexed at $a \in \mathbb{R}$ as $\mathbb{I}_a$, this interval module is given by

$$\mathbb{I}_a^{[b,d)} = \begin{cases} \mathbb{Z}_2 & \text{if } a \in [b, d) \\ 0 & \text{otherwise} \end{cases}$$

together with identity maps $\mathrm{id}_{a,a'} : \mathbb{I}_a^{[b,d)} \to \mathbb{I}_{a'}^{[b,d)}$ for all $a, a' \in [b, d)$ with $a \le a'$.

It is known due to a result of Gabriel [21] that a persistence module defined with finite complexes admits a decomposition

$$H_p\mathcal{F} \cong \bigoplus_{\alpha} \mathbb{I}^{[b_\alpha, d_\alpha)}$$

which is unique up to isomorphism and permutation of the intervals. The intervals $[b_\alpha, d_\alpha)$ are called the *bars*. The multiset of bars forms the *barcode* of the persistence module $H_p\mathcal{F}$, denoted by $\mathcal{B}_p(\mathcal{F})$. The following two definitions are taken from [14].

Definition 1. For an interval $[b, d)$, we say that ζ is a *representative cycle* for $[b, d)$, or simply ζ *represents* $[b, d)$, if one of the following holds:

- $d \neq +\infty$, ζ is a cycle in K_b containing σ_b, and ζ is not a boundary in K_{d-1} but becomes one in K_d.
- $d = +\infty$, and ζ is a cycle in K_b containing σ_b.

Definition 2 (Persistent cycles)**.** A p-cycle ζ that represents an interval $[b, d) \in \mathcal{B}_p(\mathcal{F})$ is called a *persistent p-cycle* for $[b, d)$.

For a bar $[b, d)$, σ_b is said to be a *creator simplex* and σ_d is called a *destroyer simplex*.

It is easy to check that if ζ is a representative cycle for $[b_i, d_i)$ and ξ is a representative cycle for $[b_j, d_j)$, where $b_j < b_i$ and $d_j < d_i$, then $\zeta + \xi$ is also a representative cycle for $[b_i, d_i)$. The set of representative cycles for interval $[b_i, d_i)$ is denoted by $\mathcal{R}([b_i, d_i))$. Representatives of bars of the form $[b, \infty)$ are called *essential cycles*.

Definition 3 (Persistent basis)**.** Let J be the indexing set for the intervals in the barcode $\mathcal{B}_p(\mathcal{F})$ of filtration $\mathcal{F}$. That is, for every $j \in J$, $[b_j, d_j)$ is an interval in $\mathcal{B}_p(\mathcal{F})$. Then a set of p-cycles $\{\zeta_j \mid j \in J\}$ is called a *persistent p-basis* for $\mathcal{F}$ if

$$H_p\mathcal{F} = \bigoplus_{j \in J} \mathbb{I}^{\zeta_j} \text{ where } \mathbb{I}^{\zeta_j} \text{ is defined by}$$

$$\mathbb{I}_a^{\zeta_j} = \begin{cases} [\zeta_j] & \text{if } a \in [b_j, d_j) \\ 0 & \text{otherwise.} \end{cases}$$

Here, for every $j \in J$ and every $a, a' \in [b_j, d_j)$ with $a \le a'$ the maps $\mathbb{I}_a^{\zeta_j} \to \mathbb{I}_{a'}^{\zeta_j}$ are the induced maps on homology restricted to $[\zeta_j]$, respectively.

The following theorem by Dey et al. [14] relates persistent cycles to persistent bases.

Theorem 1 ([14, Theorem 1])**.** *Let J be the indexing set for the intervals in the barcode $\mathcal{B}_p(\mathcal{F})$ of filtration $\mathcal{F}$. Then, an indexed set of p-cycles $\{\zeta_j \mid j \in J\}$ is a persistent p-basis for a filtration $\mathcal{F}$ if and only if $\zeta_j \in \mathcal{R}([b_j, d_j))$ for every $j \in J$.*

3 The ℓ_2–radius metric

Given a complex K, let $Z_p(\mathsf{K})$ denote its p-th cycle group, $B_p(\mathsf{K})$ its p-th boundary group and $H_p(K)$ its p-th homology group with $\mathbb{Z}_2$ coefficients. Given a cycle ζ, our goal is to define a non-negative weight function $w : Z_p(\mathsf{K}) \to \mathbb{R}^+$ on the cycles in $Z_p(\mathsf{K})$ and compute a minimum-weight (optimal) cycle ζ^* in its homology class $[\zeta]$, that is,

$$\zeta^* \in \arg\min_{\hat{\zeta}\in[\zeta]} w(\hat{\zeta}). \tag{1}$$

We show that the ℓ_2–radius is an alternative natural geometric objective function defined on cycles that guarantees tractability. Let P be the vertex set of K. Then, K_V denotes the subcomplex of K induced by a subset $V \subseteq P$. Extending this notation, we say a complex is *induced* by a sphere $S_{c,r}$ if it is induced by the subset of vertices of P that are enclosed by $S_{c,r}$ (including on the sphere). Let the complex induced by $S_{c,r}$ be denoted as $\mathsf{K}_{c,r}$. We define a weight function $r : C_p(\mathsf{K}) \to \mathbb{R}^+$, $\xi \mapsto r(\xi)$ where

$$r(\xi) = \min_{c,\delta}\{\delta \,|\, \xi \in C_p(\mathsf{K}_{c,\delta})\}. \tag{2}$$

In words, $r(\xi)$ is the radius of the smallest Euclidean sphere whose induced complex in K contains ξ.

We now define an ℓ_2–radius measure for intervals in a barcode. For an interval $[b, d) \in \mathcal{B}_p(\mathcal{F})$, we define $r([b, d))$ as the radius of the smallest sphere that encloses a subset of vertices V of K_b that induces a subcomplex $\mathsf{K}_b^V \subset \mathsf{K}_b$, which supports a representative cycle for $[b, d)$. Equivalently,

$$r([b, d)) = \min_{\zeta\in\mathcal{R}([b,d))} r(\zeta), \tag{3}$$

where in Equation (3), the radius function r is restricted to the subcomplex $\mathsf{K}_b \subset \mathsf{K}$.

4 Computing optimal homologous cycle

Following Equation (1), we define an optimal cycle ζ^* in the class $[\zeta]$ by requiring $\zeta^* \in \arg\min_{\hat{\zeta}\in[\zeta]} r(\hat{\zeta})$. The cycle ζ^* represents an optimal localization of the class $[\zeta]$ with respect to the ℓ_2–radius. We consider the following Optimal Homologous Cycle problem:

> Given an p-cycle $\zeta \in Z_p(\mathsf{K})$, compute an optimal cycle ζ^* in $[\zeta]$ and $r(\zeta^*)$.

Remark 4.1. To compute the optimal homologous cycle, it is sufficient to look at the minimum circumspheres of all k-subsets of points $P = V(\mathsf{K})$, where $k \in \{2, \ldots, d + 1\}$, and check if the circumsphere encloses a cycle homologous to the input cycle. When the dimension d of the complex K is fixed, the search terminates in polynomial time. This describes a trivial exact algorithm which was found to be too expensive in our experiments in spite of polynomial time complexity.

Remark 4.2. By restricting the centers of the spheres in Equation (1) to the the sites $P = V(\mathsf{K})$ (vertices of K) yields a 2-approximation of ℓ_2–radius as follows: let S be a sphere that minimizes ℓ_2–radius of a chain ξ and let v be a vertex on S. Then, a sphere of twice the optimal radius centered at v encloses S, and therefore also encloses ξ. We define $r_c(\xi) = \min\{\delta \,|\, \xi \in C_p(\mathsf{K}_{c,\delta})\}$ and $r_P(\xi) = \min_{c\in P,\delta}\{\delta \,|\, \xi \in C_p(\mathsf{K}_{c,\delta})\}$.

Notations and Conventions. The notations and conventions described are common to all the problems in the paper. In our algorithms, a cycle (or a chain) ζ is represented by a 0–1 vector in the standard chain basis. That is, a p-cycle ζ is represented by a vector ζ where $\zeta[i] = 1$ ($\zeta[i] = 0$) if a p-simplex σ_i is (not) in the support of ζ. We often use cycle vectors of subcomplexes in computations involving cycles and boundaries of larger complexes. To ensure that we are working with vectors/matrices of the right dimensions, we make the following adjustment. For complexes $\mathsf{L} \subset \mathsf{K}$, the inclusion map $\mathsf{L} \hookrightarrow \mathsf{K}$ induces maps $Z_p(\mathsf{L}) \to Z_p(\mathsf{K})$ for every p. A cycle ξ in L is mapped to a cycle $\overline{\xi}$ in K with $\overline{\xi}[i] = \xi[i]$ for simplices $\sigma_i \in \mathsf{L}$, and $\overline{\xi}[i] = 0$ for simplices $\sigma_i \in \mathsf{K} \setminus \mathsf{L}$ (using standard chain basis). Likewise, a matrix $\mathbf{M}$ of cycle vectors of L can be treated as a matrix of cycle vectors $\overline{\mathbf{M}}$ of K by padding zeros in the rows corresponding to the simplices in $\mathsf{K} \setminus \mathsf{L}$. We call such cycle vectors $\overline{\xi}$ and matrices $\overline{\mathbf{M}}$, the *extensions* of ξ and $\mathbf{M}$ in K.

Let K be a simplicial complex, $\mathsf{K} \subset \mathbb{R}^d$. For any $v \in \mathbb{R}^d$ we can define a total ordering $\prec_v$ on the simplices of K as follows. If σ_1 is a face of σ_2 or $r_v(\sigma_1) < r_v(\sigma_2)$, then $\sigma_1 \prec_v \sigma_2$. Otherwise (when $r_v(\sigma_1) = r_v(\sigma_2)$ and σ_1 is neither a face or coface of σ_2), ties are arbitrarily broken. If $\zeta = \sigma_1 + \ldots + \sigma_s$ such that $\sigma_1 \prec_v \ldots \prec_v \sigma_s$, then we define $\kappa(\zeta) = \sigma_s$. Further, we extend this ordering to chains as follows: If $\zeta_1, \zeta_2 \in C_p(\mathsf{K})$ such that $\zeta_1 = \sigma_1 + \ldots + \sigma_s$, $\zeta_2 = \sigma'_1 + \ldots + \sigma'_{s'}$ with $\sigma_1 \prec_v \ldots \prec_v \sigma_s$ and $\sigma'_1 \prec_v \ldots \prec_v \sigma'_{s'}$, then $\zeta_1 \prec_v \zeta_2$ if $\sigma_s \prec_v \sigma'_{s'}$ i.e. $\kappa(\zeta_1) \prec_v \kappa(\zeta_2)$. Note that $r_v(\zeta) = r_v(\kappa(\zeta))$. The ordering $\prec_v$ induces a simplex-wise filtration on K which we denote by $\mathcal{D}_v(\mathsf{K})$.

The standard reduction algorithm [3] is used in many of our algorithms as subroutines. For completeness, we present an outline of the algorithm and recall some facts arising out of it in Appendix C. Algorithm 1 relies on the following proposition (Proof in Appendix C).

Proposition 4. *Let $\zeta_1 \prec_v \ldots \prec_v \zeta_s$ be the essential $p-$cycles of $\mathcal{D}_v(\mathsf{K})$ computed using standard reduction. Let ζ* be a p-cycle, *$[\zeta] \neq 0 \in H_p(\mathsf{K})$, such that $\zeta = \zeta_{i_1} +$*

$\ldots + \zeta_{i_m} + \partial c_{p+1}$ *where each* $\zeta_{i_k} \in \{\zeta_1, \ldots, \zeta_s\}$ *and* c_{p+1} *is a* $p+1$ *chain. If* $i_1 < \ldots < i_m$ *then* $r_v([\zeta]) = r_v(\zeta_{i_m})$. *In particular,* $\zeta_{i_1} + \ldots + \zeta_{i_m} \in \arg\min_{\xi \in [\zeta]} r_v(\xi)$.

We now describe a 2-approximation algorithm for Optimal Homologous Cycle for an input cycle ζ by optimizing with respect to r_P. For each site v, the algorithm invokes the subroutine optimal-hom-cycle-forsite (Line 15 of Algorithm 1) which computes $r_v([\zeta]) = \min_{\eta \in [\zeta]}\{r_v(\eta)\}$ and $\zeta_v^* \in \arg\min_{\eta \in [\zeta]}\{r_v(\eta)\}$. Finally it reports the minimum among all sites and the corresponding optimal homologous cycle. Procedure optimal-hom-cycle-forsite is motivated by Proposition 4. It first sorts the simplices of K based on distance from v. The ordering is monotonic, that is, faces gain precedence over cofaces.

In this way the ordering $\prec_v$ and hence the filtration $\mathcal{D}_v(\mathsf{K})$ is defined. Let $\zeta_1 \prec_v \ldots \prec_v \zeta_m$ be the essential $p-$cycles of $\mathcal{D}_v(\mathsf{K})$ computed using standard reduction. As noted before we consider cycle vectors to represent the cycles. To compute the linear combination of cycles $\{\zeta_i\}$ which is homologous to ζ, we solve for the system of equations $[\zeta_1 \ldots \zeta_s \,|\, B_p(\mathsf{K})].x = \zeta$. (We invoke subroutine SolveByReduction which solves $Ax = b$ over $\mathbb{Z}_2$, using standard reduction as a subroutine. Refer to Appendix C, Algorithm 5 for definition of this routine). If $i_1, \ldots i_s, j_1 \ldots, j_t$ is a solution where indices $i_1, \ldots i_s$ correspond to cycles in $\{\zeta_i\}_{i=1}^m$) and $j_1 \ldots j_t$ correspond to boundaries in B_p, then by Proposition 4 $\zeta_{i_1} + \ldots + \zeta_{i_s} \in \arg\min_{\eta \in [\zeta]\{r_v(\eta)\}}$.

Remark 4.3. Algorithm 1 runs in $O(|P|N^3)$, where N is the number of simplices in K.

Algorithm 1: Computing optimal homologous cycle for given set of sites

Input : $\mathsf{K}, \zeta \in C_p(\mathsf{K})$
Output: $r_P([\zeta]), \zeta^*$ (Optimal homologous cycle)

1 **Procedure** `OptHomologousCycle`
2 $\quad r_P([\zeta]) \leftarrow \infty, \zeta^* \leftarrow \emptyset$
3 $\quad$ **for** $v \in P$ **do**
4 $\quad\quad r_v([\zeta]), \zeta_v^* \leftarrow$ Optimal-Hom-Cycle-ForSite(v)
5 $\quad\quad$ If $r_v([\zeta])$ is less than the current value of $r_P([\zeta])$, then update $r_P([\zeta])$ with $r_v([\zeta])$ and ζ^* with ζ_v^*
6 **Procedure** `Optimal-Hom-Cycle-ForSite(`v`)`
7 $\quad\quad \triangleright$ Description: Computes $r_v([\zeta]) = \min_{\eta \in [\zeta]} r_v(\eta)$, $\zeta_v^* \in \arg\min_{\eta \in [\zeta]} r_v(\eta)$
8 $\quad\quad$ Define $\prec_v$ on K. Compute $\mathcal{D}_v(\mathsf{K})$
9 $\quad\quad$ Compute the essential cycles of $\mathcal{D}_v(\mathsf{K})$ by standard reduction. Let $\zeta_1, ..., \zeta_m$ be essential cycles ordered with respect to $\prec_v$.
10 $\quad\quad$ Compute the p^{th}boundary matrix of K, denote it by B_p.
11 $\quad\quad$ Assemble matrix $\partial = [\zeta_1, ..., \zeta_m \,|\, B_p]$
12 $\quad\quad$ Solve $\partial.x = \zeta$. Invokes(SolveByReduction(∂, ζ)). Let $i_1, \ldots, i_s, j_1, \ldots, j_t$ be the solution where $i_1, \ldots, i_s \leq m$ (indices that correspond to cycles in $\{\zeta_i\}_{i=1}^m$) and $j_1 \ldots j_t > m$ (indices correspond to boundaries in B_p.)
13 $\quad\quad \zeta_v^* \leftarrow \zeta_{i_1} + \ldots + \zeta_{i_s}$.
14 $\quad\quad r_v([\zeta]) \leftarrow r_v(\zeta_v^*)$
15 $\quad\quad$ Return $r_v([\zeta]), \zeta_v^*$

5 Optimal homology basis

A set of p-cycles $\{\zeta_1, \ldots, \zeta_\beta\}$ ($\beta_p = \beta_p(\mathsf{K})$) is called a *homology cycle basis* if the set of classes $\{[\zeta_1], \ldots, [\zeta_\beta]\}$ forms a basis for $H_p(\mathsf{K})$. For simplicity, we use the term *homology basis* to refer to the set of cycles $\{\zeta_1, \ldots, \zeta_\beta\}$.

Definition 5. A $p-$homology basis $\{\zeta_1, \ldots, \zeta_m\}$ will be called a minimum $p-$homology basis ($p > 0$) with respect to a non-negative weight function $w : Z_p(K) \to \mathbb{R}$, if for all $p-$homology bases $\zeta_1' \ldots \zeta_m'$, $w([\zeta_1]) + \ldots + w([\zeta_m]) \leq w([\zeta_1']) + \ldots + w([\zeta_m'])$ and each $\zeta_i \in \arg\min_{\eta \in [\zeta_i]} w(\eta)$

We consider the following Optimal Homology Basis problem.

> For a given $p > 0$ compute a minimum $p-$homology basis with respect to the weight function r.

Algorithm 2 describes a $2\beta_p$-approximation algorithm for Optimal Homology Basis by computing a minimum homology basis with respect to r_P by restricting the centers of minimal spheres to sites. To compute the minimum homology basis $\mathscr{M}$ from Ω, (see Line 5) standard reduction is performed on $\partial = [B_p(\mathsf{K}) \,|\, \Omega]$. We examine the columns of the reduced matrix $\tilde{\partial}$ from left to right. For every non-zero column i that is an index from Ω, we add the corresponding cycle in Ω to $\mathscr{M}$. *Algorithm* 2 runs in $O(|P|N^3)$. See Appendix C.1 for a proof of correctness.

6 Optimal persistent homology basis

We now consider a filtration of a simplicial complex K with the aim of studying an extension of the problem to persistent homology [19]. We introduce the Minimum persistent homology Basis problem:

> Given a filtration $\mathcal{F}$ of complex K, compute a persistent p-basis $\Lambda_p = \{\zeta_i \mid i \in [|\mathcal{B}_p(\mathcal{F})|]\}$ that

Algorithm 2: Optimal homology basis for sites

Input : Complex $\mathsf{K} \subset \mathbb{R}^d$, $p > 0$
Output: A minimum homology basis with respect to r_P

1 **Procedure** `Opt-hom-basis-for-sites(`K, p`)`
2 For each $v \in P$, define $\prec_v$. Using standard reduction compute the essential $p-$cycles of the filtration $\mathcal{D}_v(\mathsf{K})$, denote them by $\zeta_{v,1}, \ldots, \zeta_{v,m}$.
3 Let $\Omega = \{\zeta_{v,i}\}_{v \in P, 1 \le i \le m}$. Sort the cycles in Ω so that if $r_v(\zeta_{v,i}) < r_{v'}(\zeta_{v',i'})$ then $\zeta_{v,i}$ precedes $\zeta_{v',i'}$ in Ω. If $\zeta_{v,i} \prec_v \zeta_{v,i'}$, then $\zeta_{v,i}$ precedes $\zeta_{v,i'}$ as well. Ties are broken arbitrarily. Denote this ordering on Ω as $\prec_\Omega$.
4 $\mathscr{M} \leftarrow \emptyset$.
5 **for** ζ in the ordered list Ω **do**
6 Let $\eta_1, \ldots \eta_k$ be the cycles currently in $\mathscr{M}$.
7 **if** $[\zeta] \in span\{[\eta_1], \ldots [\eta_k]\}$ **then**
8 Discard ζ and continue.
9 **else**
10 Add ζ to $\mathscr{M}$.
11 Report $\mathscr{M}$ as a minimum homology basis.

minimizes $r(\Lambda_p) = \sum_{i=1}^{|\mathcal{B}_p(\mathcal{F})|} r(\zeta_i)$.

Theorem 1 states that for computing an optimal persistent homology basis it suffices to compute the minimum representative of each bar. Formally, an optimum representative of a bar $[b, d)$ is a cycle $\zeta^* \in \arg\min_{\eta \in \mathcal{R}([b,d))}\{r_P(\eta)\}$.

Algorithm 3 computes an minimum representative of a input bar $[b, d) \in \mathcal{B}_p(\mathcal{F})$ for a simplex-wise filtration $\mathcal{F}$ of K with $V(\mathsf{K}) = P$ with respect to r_P. For each site $v \in P$ the subroutine Opt-Pers-Cycle-Site is invoked which computes $\zeta_v^* \in \arg\min_{\eta \in \mathcal{R}([b,d))}\{r_v(\eta)\}$. Finally the minimum $\zeta_P^* = \arg\min_{v \in P}\{r_v(\zeta_v^*)\}$ among all sites is reported. Similar to Algorithm 1 a filtration $\mathcal{D}_v$ is defined on K_b. The essential $p-$cycles $Y = \{\zeta_1 \prec_v \ldots \prec_v \zeta_m\}$ of $\mathcal{D}_v(\mathsf{K}_b)$ are computed using standard reduction. We then compute the smallest $i > 0$ such that $\exists \xi \in span\{\zeta_1, \ldots, \zeta_i\}, \xi \in \mathcal{R}([b, d))$. If σ_b was added at index b of $\mathcal{F}$ and α is the index of the first cycle in Y containing σ_b, then update Y by adding Y_α to all other cycles containing σ_b. This ensures that only a single cycle now contains σ_b. Denoting these cycles of $Y \setminus \{\zeta_\alpha\}$ by Y' and the first i cycles of Y' by $Y'_{\le i}$, it suffices to check if $[B_p(\mathsf{K}_d) \,|\, Y_{\le i}].x = Y_\alpha$ has a solution. This is determined in Line 15 with a binary search over $i \in [1..m-1]$.

The proof of correctness of Algorithm 3 can be found in Appendix C.2. It runs in $O(|P|N^3 \log N)$.

Algorithm 3: Computing optimal representative of bar of persistence wrt r_P.

Input : $\mathsf{K}, \mathcal{F}$(simplex-wise filtration), $[b, d) \in \mathcal{B}_p(\mathcal{F})$
Output: $\zeta_P^*([b, d)) \in \arg\min_{v \in P, \eta \in \mathcal{R}([b,d))}\{r_v(\eta)\}$

1 **Procedure** `Opt-PersHom-Rep(`$\mathsf{K}, \mathcal{F}, [b, d)$`)`
2 $r_P([b, d)) \leftarrow \infty, \zeta_P^*([b, d)) \leftarrow \emptyset$
3 **for** $v \in P$ **do**
4 $r_v(\zeta_v^*), \zeta_v^* \leftarrow$ Opt-Pers-Cycle-Site$([b, d), v)$
5 **if** $r_v(\zeta_v^*) < r_P([b, d])$ **then**
6 $r_P([b, d)) \leftarrow r_v(\zeta_v^*)\ \zeta_P^*([b, d)) \leftarrow \zeta_v^*$
7 Return $r_P([b, d)), \zeta_P^*([b, d))$

8 **Procedure** `Opt-Pers-Cycle-Site(`$[b, d), v$`)`
9 Define $\prec_v$ on K_b. Compute $\mathcal{D}_v(\mathsf{K}_b)$
10 Compute the essential p-cycles $\zeta_1 \prec_v \ldots \prec_v \zeta_m$ of $\mathcal{D}_v(\mathsf{K}_b)$ using standard reduction
11 $Y \leftarrow \zeta_1, ..., \zeta_m$. ($Y$ is a matrix of m columns, the column Y_i is the cycle-vector of ζ_i). Let α be index of first cycle in Y containing σ_b
12 Add cycle Y_α to all other cycles in Y containing σ_b, resulting in matrix $\hat{Y}$.
13 Assemble matrix Y' by dropping the α^{th} column of $\hat{Y}$. Denote by $Y'_{\le i}$ the first i columns of Y'.
14 $\partial_d \leftarrow B_p(K_d)$($\partial_d$ is empty if $d = \infty$)
15 Compute the smallest $i \in [1..m-1]$ such that $[\partial_d | Y'_{\le i}].x = Y_\alpha$ has a solution.
16 Let $b_1, ..., b_t, i_1, ..., i_s$ be the solution computed by the previous step where $b_1, ..., b_t$ are indices in ∂_d and $i_1, ..., i_s$ are in $Y'_{\le i}$
17 $\zeta_v^* \leftarrow Y'_{\le i, i_1} + ... + Y'_{\le i, i_s} + Y_\alpha$
18 Return $\zeta_v^*, r_v(\zeta_v^*)$

7 Experiments

We report results of experiments on real world datasets with a focus on computing cycle representatives of H_1. These results demonstrate the utility of the ℓ_2–radius towards the identification of meaningful representatives of homology classes. We consider three applications: localizing individual 1-cycles, optimal 1-homology basis computation, and minimum persistent 1-homology basis computation. Computing the exact ℓ_2–radius via an enumeration of all circumspheres is expensive. So, all experiments were conducted on implementations of the approximate algorithms described above.

We implement a heuristic to minimize the length of the cycle representative while preserving its radius. Essentially, we replace one of the paths P between

two vertices in the cycle with the shortest length path between them if it is homologous to P. This heuristic results in smoother and shorter cycles for all datasets. This also addresses the issue with the non-unicity of the ℓ_2-metric in the sense that in our experiments we always find tighter cycles within a sphere when there are several homologous cycles of varying lengths within a sphere.

All experiments were performed on an Intel Xeon(R) Gold 6230 CPU @ 2.10 powered workstation with 20 cores and 384GB RAM running Ubuntu Linux. The algorithms were parallelized using Intel Thread Building Blocks (TBB). The PHAT library [3] was used in all routines that invoke the standard reduction algorithm.

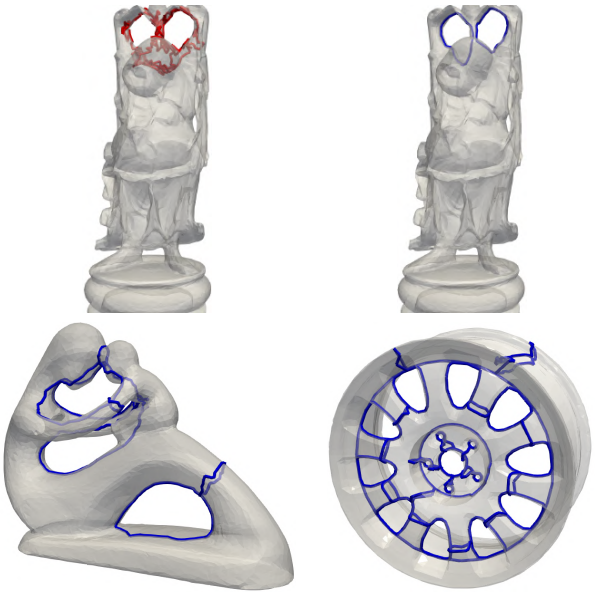
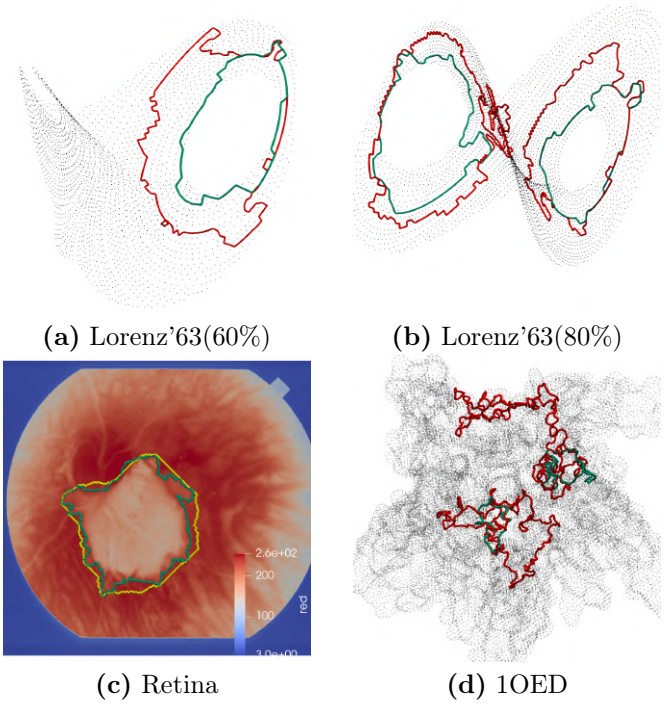


Figure 1: (**top**) The localization algorithm computes optimal (blue) 1-cycles that are homologous to input 1-cycles (red). (**bottom**) Minimum 1-homology basis.

Homology localization. Given an input simplicial complex representing a surface and a 1-cycle, we compute a localized cycle that is homologous to the input cycle. Figure 1(top) shows two input cycles (red) and their localized versions (blue) for the Happy Buddha dataset. We can visually infer that the localized cycle computed by our approximate algorithm is close to the optimal cycle.

Optimal homology basis. Figure 1(bottom) shows the optimal homology basis computed for two 3D models. We observe that the cycles are tight and capture all tunnels and loops of the model. Results on additional datasets are available in Appendix D.

Optimal persistent homology cycle. We report our results on three classes of filtrations: Rips, lower star, and Delaunay. Figure 2 **(a,b)** shows the



(a) Lorenz'63(60%) **(b)** Lorenz'63(80%)

(c) Retina **(d)** 1OED

Figure 2: The green persistent 1-homology cycles computed by our algorithm are tighter than the red (or yellow) cycles computed by PersLoop [15].

Lorenz'63 data set for 60% and 80% densities and the representatives of the longest and the top two longest bars, respectively, of a Rips filtration. These are of relevance in simulating weather phenomena [29]. Figure 2 **(c)** highlights the representative of the longest lived bar for a lower star filtration on a retinal image [23] with a retinal disorder. The cycle represents the region of the disorder. Figure 2 **(d)** highlights representatives of the top two bars of an alpha complex on a protein molecule (PDB-ID: 1OED). Cycles computed by our algorithm (green) appear "tighter" than those computed by PersLoop (red or yellow).

Execution time. Our algorithms are parallelizable if the optimization subroutines for each site can be executed independently. We obtain fast running times ($\sim$ few minutes) for moderate to large-sized datasets ($\sim$ millions of simplices) for all algorithms, see Table 1.

Data set	#Simplices	Execution time
Lorenz-63(60%)	$\sim 2 \times 10^6$	120s
Lorenz-63(80%)	$\sim 3 \times 10^6$	120s
Retina	$\sim 5 \times 10^6$	140s
1OED	$\sim 2.5 \times 10^6$	130s

Table 1: Mean time to compute optimal representative of each of the top-40 bars in the persistence barcode.

Acknowledgements

This work is partially supported by the PMRF, MoE Govt. of India, a SERB grant CRG/2021/005278, an NSF grant CCF 2049010 and ERC 101039913. VN acknowledges support from the Alexander von Humboldt Foundation, and Berlin MATH+ under the Visiting Scholar program. Part of this work was completed when VN was a guest Professor at the Zuse Institute Berlin. AR would like to thank Tamal K. Dey for several helpful discussions and for his invaluable support.

References

[1] H. Bakke Bjerkevik. On the stability of interval decomposable persistence modules. *Discrete & Computational Geometry*, 66(1):92–121, 2021.

[2] A. Barbensi, I. H. R. Yoon, C. D. Madsen, D. O. Ajayi, M. P. H. Stumpf, and H. A. Harrington. Hypergraphs for multiscale cycles in structured data, 2022.

[3] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. PHAT–persistent homology algorithms toolbox. *Journal of symbolic computation*, 78:76–90, 2017.

[4] G. Borradaile, E. W. Chambers, K. Fox, and A. Nayyeriy. Minimum cycle and homology bases of surface-embedded graphs. *Journal of Computational Geometry*, 8(2), 2017.

[5] G. Borradaile, W. Maxwell, and A. Nayyeri. Minimum bounded chains and minimum homologous chains in embedded simplicial complexes. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 21:1–21:15, 2020.

[6] M. Botnan. Mastermath course topological data analaysis. `http://www.few.vu.nl/~botnan/lecture_notes.pdf`, 2022.

[7] E. W. Chambers, J. Erickson, and A. Nayyeri. Minimum cuts and shortest homologous cycles. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 377–385. ACM, 2009.

[8] E. W. Chambers, S. Parsa, and H. Schreiber. On complexity of computing bottleneck and lexicographic optimal cycles in a homology class. In *38th International Symposium on Computational Geometry, SoCG 2022*, volume 224 of *LIPIcs*, pages 25:1–25:15, 2022.

[9] C. Chen and D. Freedman. Measuring and computing natural generators for homology groups. *Computational Geometry*, 43(2):169–181, 2010.

[10] C. Chen and D. Freedman. Hardness results for homology localization. *Discrete & Computational Geometry*, 45(3):425–448, 2011.

[11] D. Cohen-Steiner, A. Lieutier, and J. Vuillamy. Lexicographic optimal homologous chains and applications to point cloud triangulations. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 32:1–32:17, 2020.

[12] D. Cohen-Steiner, A. Lieutier, and J. Vuillamy. Lexicographic optimal homologous chains and applications to point cloud triangulations. *Discrete & Computational Geometry*, pages 1–20, 2022.

[13] T. K. Dey, A. N. Hirani, and B. Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM Journal on Computing*, 40(4):1026–1044, 2011.

[14] T. K. Dey, T. Hou, and S. Mandal. Persistent 1-cycles: Definition, computation, and its application. In *International Workshop on Computational Topology in Image Context*, pages 123–136. Springer, 2019.

[15] T. K. Dey, T. Hou, and S. Mandal. Computing minimal persistent cycles: Polynomial and hard cases. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2587–2606. SIAM, 2020.

[16] T. K. Dey, J. Sun, and Y. Wang. Approximating loops in a shortest homology basis from point data. In *Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry*, pages 166–175. ACM, 2010.

[17] T. K. Dey and Y. Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2022. `https://www.cs.purdue.edu/homes/tamaldey/book/CTDAbook/CTDAbook.pdf`.

[18] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. Applied Mathematics. American Mathematical Society, 2010.

[19] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28:511–533, 2002.

[20] J. Erickson and K. Whittlesey. Greedy optimal homotopy and homology generators. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1038–1046. Society for Industrial and Applied Mathematics, 2005.

[21] P. Gabriel. Unzerlegbare Darstellungen I. *Manuscripta Mathematica*, 6(1):71–103, 1972.

[22] G. Henselman-Petrusek. personal communication.

[23] A. Hoover and M. Goldbaum. Locating the optic nerve in a retinal image using the fuzzy convergence of the blood vessels. *IEEE Transactions on Medical Imaging*, 22(8):951–958, 2003.

[24] L. Li, C. Thompson, G. Henselman-Petrusek, C. Giusti, and L. Ziegelmeier. Minimal cycle representatives in persistent homology using linear programming: An empirical study with user's guide. *Frontiers in artificial intelligence*, 4:681117, 2021.

[25] I. Obayashi. Volume-optimal cycle: Tightest representative cycle of a generator in persistent homology. *SIAM Journal on Applied Algebra and Geometry*, 2(4):508–534, 2018.

[26] I. Obayashi. Stable volumes for persistent homology. *Journal of Applied and Computational Topology*, 7(4):671–706, 2023.

[27] A. Rathod. Fast algorithms for minimum cycle basis and minimum homology basis. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 64:1–64:11, 2020.

[28] Sayan Mandal, Tamal K Dey , Tao Hou. "persloop software for computing persistent 1-cycle. `https://github.com/Sayan-m90/Persloop-viewer`.

[29] K. Strommen, M. Chantry, J. Dorrington, and N. Otter. A topological perspective on weather regimes. *Climate Dynamics*, 60:1–31, 07 2022.

Generalizing Combinatorial Depth Measures to Line Segments*

Stephane Durocher† Myroslav Kryven†

Abstract

A data depth measure quantifies the depth of a given datum relative to a given set of data. Depth measures are important tools for statistical data inference. Common depth measures such as simplicial depth and Tukey depth have been studied extensively for sets of points in $\mathbb{R}^d$. We generalize definitions for these depth measures to the setting of sets of line segments in $\mathbb{R}^2$. That is, given a line segment q and a set S of line segments in $\mathbb{R}^2$, we seek to evaluate how deeply nested q is relative to S. In this paper, we introduce three depth measures for sets of line segments, as well as efficient algorithms for computing them.

1 Introduction

Depth measures quantify the centrality or outlyingness of an individual object relative to a population (a set, multiset, or probability distribution) of objects. Common depth measures for point data in $\mathbb{R}^d$ include simplicial depth [11] and Tukey (half-space) depth [13]. *Simplicial depth* is equal to the number of simplices (triangles in the two-dimensional case) defined on a point set that contain the query point, while *Tukey depth* is equal to the smallest number of vertices of the point set contained in any half-plane that also contains the query point. While much previous work on depth measures has focused on multivariate point data, some recent work examines generalizations to other types of data in $\mathbb{R}^d$, including curves and polylines [6, 9]. Applications include evaluating the degree of similarity of a given polyline (e.g., an individual GPS trajectory or animal migration track) relative to a given population of such polylines [9]. In this paper, we examine depth for sets of line segments in $\mathbb{R}^2$, seeking to bridge the gap between depth measures for points and depth measures for polylines and curves. Although existing depth measures for curves and polylines (e.g., curve stabbing depth [9]) can be applied to line segments, a broader range of techniques for defining and computing depth can be applied to line segments, allowing natural generalizations of previous depth measures and resulting in simpler definitions for depth measures and significantly more efficient algorithms for computing them.

The paper is organized as follows. In Section 2 we provide definitions for the combinatorial depth measures under consideration and their generalizations to segments. For each depth measure, we first present an algorithm to compute the depth of a given point q relative to a given set S of n line segments, and then give an algorithm to compute the depth of a given line segment s relative to S. In Section 3, we show how to compute the simplicial depth of s relative to S in $O(n^2)$ time using half-space counts introduced by Rousseeuw and Ruts [12] and a sweep-line technique. In Section 4, we give a randomized algorithm with expected time $O(n^{4/3} \log n)$ for computing the eutomic depth of s relative to S. Our algorithm uses a randomized algorithm of Chan [3] for constructing a k-level of a line arrangement. Finally, in Section 5, we give an optimal randomized algorithm with expected time $O(n \log n)$ to compute the Tukey depth of s relative to S using a randomized optimization technique of Chan [4].

2 Defintions

Let P be a set of points and let q be a point in $\mathbb{R}^2$. The *half-space depth* [13] (also known as *Tukey depth*) of q relative to P, denoted $\mathrm{TD}(q, P)$, is the minimum number of points of P in any closed half-plane h containing q:

$$\mathrm{TD}(q, P) = \min_{\substack{h: q \in h, \\ h \in \mathcal{H}}} |P \cap h|,$$

where $\mathcal{H}$ is the set of all closed half-planes in $\mathbb{R}^2$.

We generalize Tukey depth for segments, which we assume to be closed (that is, the endpoints are included). First, we define the Tukey depth of a query point q relative to a set S of segments in $\mathbb{R}^2$ as the minimum number of segments in S that intersect any closed half-space containing q:

$$\dot{\overline{\mathrm{TD}}}(q, S) = \min_{\substack{h: q \in h, \\ h \in \mathcal{H}}} |S \cap h|.$$

Next, we define the Tukey depth of a segment s relative to a set S of segments in $\mathbb{R}^2$ as the maximum depth $\dot{\overline{\mathrm{TD}}}(q, S)$ of any point $q \in s$ relative to S:

$$\overline{\mathrm{TD}}(s, S) = \max_{q: q \in s} \dot{\overline{\mathrm{TD}}}(q, S).$$

*This work is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†Department of Computer Science, University of Manitoba, {stephane.durocher,myroslav.kryven}@umanitoba.ca

Another classical depth measure of a point q relative to a set P of points in $\mathbb{R}^2$ is *simplicial depth* [11], denoted $\mathrm{SD}(q, P)$, which is the number of open simplices (some definitions use closed simplices) determined by points in P that contain q:

$$\mathrm{SD}(q, P) = \sum_{\{x,y,z\}\subseteq P} \mathrm{I}(q \in \triangle xyz),$$

where I is an indicator function such that $I(A) = 1$ if A is true and $I(A) = 0$ otherwise, where A is a predicate. We generalize simplicial depth for segments. As we did for Tukey depth, we first define the simplicial depth of a point q relative to a set S of line segments in $\mathbb{R}^2$ as the number triples of segments $\{s_1, s_2, s_3\} \subseteq S$ such that some simplex (triangle) with respective vertices on s_1, s_2, and s_3 contains q:

$$\dot{\overline{\mathrm{SD}}}(q, S) = \sum_{\{s_1,s_2,s_3\}\subseteq S} \max_{\substack{x\in s_1,\\ y\in s_2,\\ z\in s_3}} \mathrm{I}(q \in \triangle xyz).$$

We define simplicial depth for a segment s relative to a set S of segments in $\mathbb{R}^2$ as the maximum depth $\dot{\overline{\mathrm{SD}}}(q, S)$ of any point $q \in s$ relative to S:

$$\overline{\mathrm{SD}}(s, S) = \max_{q:q\in s} \dot{\overline{\mathrm{SD}}}(q, S).$$

Recently, a new combinatorial depth measure $\mathrm{ED}(q, P)$, called *eutomic depth* [8], was defined for a point q relative to a set P of points in $\mathbb{R}^2$ as the number of combinatorially *distinct* (that is, they induce distinct partitions of P) *halving lines* (lines that partition P into two sets whose cardinalities differ by at most one) passing through q (halving lines do not pass through any points of P), more precisely:

$$\mathrm{ED}(q, P) = \sum_{p\in P} I\left(\left|\, |H^+_{q,p} \cap P| - |H^-_{q,p} \cap P| \,\right| \leq 1\right),$$

where $H^+_{q,p}$ and $H^-_{q,p}$ denote the two open half-planes determined by the line through the points q and p.

We again first generalize eutomic depth for a point q with respect to a set S of segments. A *halving line* of S is a line having equal number of endpoints of segments of S on each side (points may not be on the line). Thus, the definition of eutomic depth for points naturally extends to segments. The *eutomic depth* $\dot{\overline{\mathrm{ED}}}(q, S)$ of a point q with respect to S is the number of combinatorially distinct halving lines passing through q, that is, we define $\dot{\overline{\mathrm{ED}}}(q, S) = \mathrm{ED}(q, P_S)$, where P_S is the set of endpoints of the segments in S. As before, we use this definition to define *eutomic depth* of a segment s with respect to S.

$$\overline{\mathrm{ED}}(s, S) = \max_{q:q\in s} \dot{\overline{\mathrm{ED}}}(q, S).$$

An object s with maximum depth (for a given depth measures) relative to a set S is a *depth median* of S. Another helpful notion, introduced by Rousseeuw and Ruts [12], for defining and measuring Tukey depth and simplicial depth of a point q relative to a set P of n points is *half-space counts*, denoted $h_i(q)$, and defined as follows. Suppose the points (in general position) in P are labelled in radial order around q so that $0 \leq \alpha_1 < \cdots < \alpha_n < 2\pi$, where α_i is the angle between the vectors $p_i - q$ and $(1, 0)$. Because the set $P \cup \{q\}$ is in general position, the angles are all unique. The *half-space count* $h_i(q)$ is the largest integer k such that $\alpha_i < \alpha_{i+1} \leq \alpha_{i+k} < \alpha_i + \pi$, where $\alpha_{n+j} = \alpha_j + 2\pi$ for all j. This definition is equivalent to counting the number of points of P in the right open half-plane $\gamma_i(q)$ defined by the line through the points (p_i, q) and the vector $p_i - q$. Rousseeuw and Ruts [12] showed that we can compute $h_i(q)$ for all i in $O(n)$ time when the points P are sorted as described above (or $O(n \log n)$ time if unsorted). Using half-space counts, we can compute simplicial depth (see also Section 3) and Tukey depth of a point relative to a set of n segments in $O(n \log n)$ time [2]. See Aloupis' survey of geometric measures of data depth [1] for further discussion.

3 Simplicial Depth

From now on we assume that our input is in general position; that is, no three input points (including segment endpoints) are collinear.

The brute-force approach for computing $\mathrm{SD}(q, P)$ for a point q and a set P of n points follows from the definition: among all triangles formed by the points in P, count those that contain q. This approach can also be used to compute $\dot{\overline{\mathrm{SD}}}(q, S)$ for the point q and a set S of n segments: among all triples of segments in S, count those that form a triangle containing q (each segment contains one triangle vertex). Given segments s_1, s_2, s_3 and a point q, we can determine in $O(1)$ time whether there exist points $x \in s_1$, $y \in s_2$, and $z \in s_3$ such that $q \in \triangle xyz$. Simplicial depth for points can be computed faster using half-space counts [2]:

$$\mathrm{SD}(q, P) = \binom{n}{3} - \sum_{i=1}^{n} \binom{h_i(q)}{2}, \tag{1}$$

where the second term counts the triangles that do not contain the point q; $\binom{h_i(q)}{2}$ counts triangles $\Delta p_i p_{i_1} p_{i_2}$ such that p_{i_1} and p_{i_2} are in the half-plane $\gamma_i(q)$ corresponding to $h_i(q)$. Note that we do not double count as the sets of triangles $\Delta p_i p_{i_1} p_{i_2}$ for each i are disjoint. Rousseeuw and Ruts [12] showed that we can compute $h_i(q)$ for all i in time $O(n \log n)$ by first sorting the data points radially (in counter-clockwise order) around q as described above and then performing a radial sweep.

Computing $\dot{\overline{\mathrm{SD}}}(q, S)$ We can use the approach of Rousseeuw and Ruts [12] to compute $\dot{\overline{\mathrm{SD}}}(q, S)$ in time $O(n \log n)$ as follows. First, we sort the segments in S radially counter-clockwise around q (in the order in which the segments are first met during the radial sweep). Then for each segment s_i in this order, let $\gamma_i(q)$ be the closed half-plane through q and one of the endpoints of s_i, such that $\gamma_i(q)$ contains the entire segment s_i (this avoids double counting in Equation (2)). Note that no triangle formed by s_i and any other two segments entirely contained in $\gamma_i(q)$ can contain q in its interior. Therefore, we can use the same idea as in (1) to compute $\dot{\overline{\mathrm{SD}}}(q, S)$:

$$\dot{\overline{\mathrm{SD}}}(q, S) = \binom{n}{3} - \sum_{i=1}^{n} \binom{\bar{h}_i(q)}{2}, \tag{2}$$

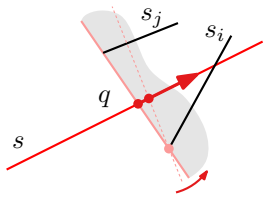
where $\bar{h}_i(q)$ is the number of segments entirely contained in $\gamma_i(q)$; $\binom{\bar{h}_i(q)}{2}$ counts triples of segments (s_i, s_{i_1}, s_{i_2}) such that $s_{i_1}, s_{i_2} \in \gamma_i(q)$. Again there is no double counting because the sets of triples (s_i, s_{i_1}, s_{i_2}) for each fixed i are disjoint. Using the approach of Rousseeuw and Ruts [12], we can also compute $\bar{h}_i(q)$ for all i in time $O(n \log n)$ by sorting the segments by their endpoints radially around q and then performing a radial sweep.

Computing $\overline{\mathrm{SD}}(s, S)$ Let s be a segment and let S be a set of n segments. Recall that $\overline{\mathrm{SD}}(s, S)$ is the depth of a point $q \in s$ that maximizes $\dot{\overline{\mathrm{SD}}}(q, S)$. Simplicial depth does not have convex depth contours (in particular, point sets where the deepest points form several disjoint clusters are known [8]). Consequently, it is unlikely that the $O(n \log n)$-time algorithm for computing $\dot{\overline{\mathrm{SD}}}(q, S)$ can be used to compute $\overline{\mathrm{SD}}(s, S)$ in $O(n \operatorname{polylog}(n))$ time by using some parametric search technique (e.g., binary search). Below we prove that we can compute $\overline{\mathrm{SD}}(s, S)$ in $O(n^2)$ time.

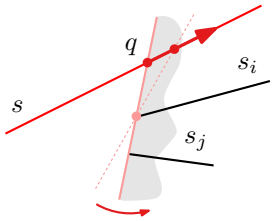
Theorem 1 *For a segment s and set S of n segments, we can compute $\overline{\mathrm{SD}}(s, S)$ in time $O(n^2)$.*

Proof. We sweep the segment $s = \overline{ab}$ from one endpoint to the other, show that we only need to consider $O(n^2)$ discrete event points, and show how to update $\dot{\overline{\mathrm{SD}}}(q, S)$ at each event point in $O(1)$ time.

First we compute $\dot{\overline{\mathrm{SD}}}(q, S)$ at one of the endpoints of s, say for $q = a$, using the approach of Rousseeuw and Ruts [12] as described above. Then we show how to update $\dot{\overline{\mathrm{SD}}}(q, S)$ as q moves along s from a to b. Observe that as we move the point q along the segment s, $\dot{\overline{\mathrm{SD}}}(q, S)$ changes only if some half-space count $\bar{h}_i(q)$ (for the endpoints of the segments) changes. A half-space count $\bar{h}_i(q)$ changes only if the radial order of the endpoints of the segments in S changes with respect to



(a) s_j has started leaving $\gamma_i(q)$



(b) s_j has finished entering $\gamma_i(q)$

Figure 1: Illustration for the proof of Theorem 1.

q. That happens when q passes the intersection point of the segment s and some line through the endpoints of two segments in S. Denote such an event $q(x_i, x_j)$, where x_i is an endpoint of the segment $s_i \in S$ and x_j is an endpoint of the segment $s_j \in S$, $s_i \neq s_j$, and q crosses the line through x_i and x_j. Because our input is in general position, the only half-space counts that may change at the event $q(x_i, x_j)$ are $\bar{h}_i(q)$ and $\bar{h}_j(q)$. More precisely, if the segment s_j has started leaving (see Figure 1a) the half-plane $\gamma_i(q)$, then we decrement $\bar{h}_i(q)$ by one; if the segment s_j has finished entering (see Figure 1b) $\gamma_i(q)$, then we increment $\bar{h}_i(q)$ by one. Therefore, we can update $\bar{h}_i(q)$ and $\bar{h}_j(q)$ and consecutively $\dot{\overline{\mathrm{SD}}}(q, S)$ in $O(1)$ time. Finally, we report the maximum value of $\dot{\overline{\mathrm{SD}}}(q, S)$ reached at an event $q(x_i, x_j)$. Because there are at most $O(n^2)$ such events and we can update $\dot{\overline{\mathrm{SD}}}(q, S)$ at each event in $O(1)$ time, the total runtime of our algorithm is $O(n^2)$. □

4 Eutomic Depth

As is the case for the simplicial depth and Tukey depth, the eutomic depth of a point q relative to a set P of n points in $\mathbb{R}^2$ can be computed in $O(n \log n)$ time [8] using half-space counts. For any point q and any set S of n segments, $\dot{\overline{\mathrm{ED}}}(q, S) = \mathrm{ED}(q, P_S)$, where P_S is the set of endpoints of the segments in S. Consequently, we can compute $\dot{\overline{\mathrm{ED}}}(q, S)$ in time $O(n \log n)$. Durocher et al. [8] also described an $O(n^{8/3})$-time algorithm to find a eutomic median (that is, a deepest point with respect to eutomic depth). Their algorithm works in three steps: First, they construct all halving lines through every two input points in time $O(n^{4/3})$ [7]. Second, they analyze $O(n^{8/3})$ many cells in the resulting line arrangement;

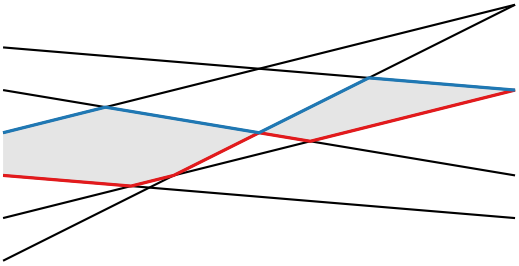


Figure 2: Illustration of median corridor for $n = 3$; the lower 3-level is in red and the upper 3-level is in blue.

for each cell, each point q in the cell has the same number of combinatorially distinct halving lines through q, and thus, the same eutomic depth. Finally, they traverse each cell of the arrangement to find a point of maximum depth. Inspired by their application of the result of Dey [7], we apply a similar technique to compute $\overline{\mathrm{ED}}(q, S)$. Instead of constructing an arrangement of halving lines, we switch to dual space [10], where our input n segments become a set of pairs of lines, each pair representing a segment. We construct an *n-level*, that is, the curve consisting of the points that lie on one of the $2n$ lines and have exactly $n-1$ lines strictly above. We call this an *upper n-level*. We also construct a *lower n-level* which is defined symmetrically, that is, the polyline consisting of the points that lie on one of the $2n$ lines and have exactly $n-1$ lines strictly below. A k-level of an arrangement of x-monotone curves is an x-monotone polygonal chain on the arrangement that switches curves at every vertex of the arrangement it encounters [3]. We call the closed region bounded by the lower n-level and the upper n-level the *median corridor*. The *size* of a k-level is the number of vertices that it contains. Dey [7] showed that it is of order $O(nk^{1/3})$ for n lines; therefore, the size of our median corridor is $O(n^{4/3})$. The *complexity* of the median corridor is the number of cells from the arrangement of the $2n$ lines that it contains relative to its size.

Observation 1 *For an arrangement of $2n$ lines the complexity of median corridor is $O(n^{4/3})$.*

The above observation easily follows from the size of the upper and lower n-levels [7] and the fact that median corridor does not have any intersections of the lines strictly inside. Moreover, because the number of lines is even, the median corridor is a chain consisting of the *cells* (that is, connected regions after removing the lines) of the arrangement. In other words, the cells of the median corridor form a path where two consecutive cells share a crossing point; see Figure 2.

We show that we can compute $\overline{\mathrm{ED}}(s, S)$ in $O(n^{4/3})$ expected time using Chan's [3] randomized algorithm for constructing k-level.

Theorem 2 *For a segment s and a set of n segments S we can compute $\overline{\mathrm{ED}}(s, S)$ in $O(n^{4/3} \log n)$ expected time.*

Proof. First we translate and rotate the input segments so that the segment s becomes vertical and one of its endpoints matches the origin of the coordinate system. Then we switch to dual space where our input set S becomes a set of $2n$ lines. We construct a median corridor using Chan's [3] algorithm in $O(n^{4/3})$ expected time. Recall that to compute $\overline{\mathrm{ED}}(s, S)$ we need to find a point q on s that maximizes $\overline{\mathrm{ED}}(q, S)$. Because our segment s is vertical with one endpoint at the center of the coordinate system, any point on the segment corresponds to a horizontal line in the dual space. Therefore, in the primal space, a point on the segment that maximizes eutomic depth $\overline{\mathrm{ED}}(q, S)$, that is, a point that has the largest number of combinatorially different halving lines corresponds, in the dual space, to a horizontal line that intersects the maximum number of *cells* (connected regions after removing the lines) of the median corridor. Because the complexity of the median corridor is linear in its size $O(n^{4/3})$ (in particular, the median corridor does not have any line intersections strictly inside, see Observation 1), we can project the cells of the median corridor on the vertical line by projecting each point of the cell onto the vertical line. Then the problem reduces to finding a point contained in the maximum number of 1-dimensional intervals (or finding maximal clique of the resulting interval graph). There are $O(n^{4/3})$ such intervals, and thus, after sorting them we can find the maximum number of intervals containing the same point in time $O(n^{4/3} \log n)$.

Therefore, the total expected runtime is $O(n^{4/3} \log n)$.

□

5 Tukey depth

Unlike eutomic depth, the Tukey depth of a point q relative to a set S of n line segments cannot be expressed in terms of Tukey depth relative to the set of endpoints of segments in S; this is because for each half-plane h in the computation of $\overline{\mathrm{TD}}(q, S)$, each segment contributes one unit to the depth regardless whether it is completely contained in h or whether it only partially intersects h. We show, however, that we can apply a randomized optimization technique developed by Chan [4] to compute both $\overline{\mathrm{TD}}(q, S)$ and $\overline{\mathrm{TD}}(s, S)$ for a segment s in optimal expected time $O(n \log n)$.

Chan [4] gave a randomized optimization technique for a class of optimization problems, called *LP-type* problems (see Definition 1) that share properties with linear and convex programming [4]. Many geometric optimization problems can be reduced to such LP-

type problems, including answering linear programming queries, finding the minimum diameter of moving points, inverse parametric minimum spanning trees, and, in particular, finding a Tukey median of a set of points. We show that this approach also extends to computing $\overline{\dot{\mathrm{TD}}}(q, S)$ as well as $\overline{\mathrm{TD}}(s, S)$; in particular, we show that both problems can be expressed as LP-type problems with recursive substructure (see Lemma 3) so that the randomized technique of Chan can be applied to get an $O(n \log n)$-time randomized algorithm.

Definition 1 *Let $w : 2^{\mathcal{H}} \to \mathcal{W}$ be a function that maps sets of* constraints *(members of $\mathcal{H}$) to values in a totally ordered set $\mathcal{W}$. We say that w is* LP-type *of dimension at most d if the following properties hold for all sets $H \subseteq \mathcal{H}$ and all constraints $h \in \mathcal{H}$:*

1. *$w(H) = w(B)$ for some $B \subseteq H$ of size at most d.*
2. *$w(H \cup \{h\}) \geq w(H)$.*
3. *Suppose $w(H) = w(B)$ with $B \subseteq H$. Then $w(H \cup \{h\}) = w(H) \Leftrightarrow w(B \cup \{h\}) = w(B)$.*

A set B of size d is called a *basis*, and if (1) holds, a *basis for* H. If $w(B \cup \{h\}) = w(B)$, we say B satisfies h and if $w(B \cup H) = w(B)$, we say B satisfies H. The algorithms solving LP-type problems may use some primitive operations, such as *basis evaluation* (computing $w(B)$ for a basis B) and *satisfaction/violation test*, that is, determining if a basis B satisfies or violates constraint h.

Chan [4] proved the following lemma.

Lemma 3 ((Chan 2004) Lemma 2.3 [4]) *Let $w : 2^{\mathcal{H}} \to \mathcal{W}$ be an LP-type function of constant dimension d and let $\alpha < 1$ and r be fixed constants. Suppose $f : \mathcal{P} \to 2^{\mathcal{H}}$ is a function that maps inputs to sets of constraints, with the following properties:*

0. *For inputs $P_1, \ldots, P_d \in \mathcal{P}$ of constant size, a basis for $f(P_1) \cup \cdots \cup f(P_d)$ can be computed in $O(1)$ time.*
1. *For any input $P \in \mathcal{P}$ and any basis B, we can decide whether B satisfies $f(P)$ in time $D(n)$.*
2. *For inputs $P \in \mathcal{P}$, we can construct inputs $P_1, \ldots, P_r \in \mathcal{P}$ each of size at most $\lceil \alpha n \rceil$, in time no more than $D(n)$, such that*

$$f(P) = f(P_1) \cup \cdots \cup f(P_r).$$

Then we can compute a basis for $f(P)$ in $O(D(n))$ expected time.

Let s be a segment and let S be a set of n segments. We first study the problem of finding a point on s with Tukey depth at least k. This problem is closely related to linear programming because we are checking whether the following intersection of half-planes

$$\bigcap \{\gamma : \text{over all half-planes } \gamma \text{ with } |S \setminus \gamma| < k\} \qquad (3)$$

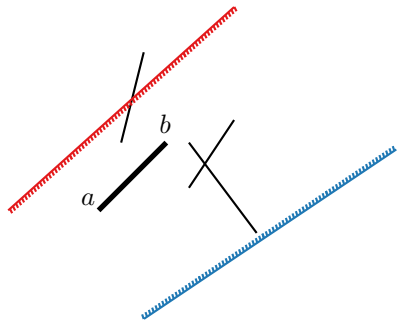
together with the segment s is not empty. In our further discussion it is helpful to switch to dual space [10]. Each segment $\overline{ab} \in S$ becomes in the dual space a pair of lines $a^\star$ and $b^\star$ that we call a *wedge*. A point q on the segment ab corresponds in the dual space to a line $h = q^\star$, passing through the intersection of the *upper half-plane* defined by $a^\star$ (the half-plane above the line $a^\star$) and the *lower half-plane* defined by $b^\star$, as well as the intersection of $a^\star$ and $b^\star$; we say that h is a line *of the cone* formed by $a^\star$ and $b^\star$.

For each segment $\overline{ab} \in S$, let $H := \{(a^\star, b^\star) : ab \in S\}$. When the segment $\overline{ab}$ intersects an upper half-plane defined by h (in other words, one of the endpoints a or b is above the line of h), the point $h^\star$ is above one of the lines $a^\star$ and $b^\star$. In this case, we say the point $h^\star$ is above the *lower v-shaped ray* formed by the two lines $a^\star$ and $b^\star$. Similarly, when the segment $\overline{ab}$ intersects a lower half-plane defined by h, we say that the point $h^\star$ is below the *upper v-shaped ray* formed by $a^\star$ and $b^\star$; see Figure 3. For the pairs of lines in H, denote the set of upper v-shaped rays V, the set of lower v-shaped rays Λ, and let $W = V \cup \Lambda$.

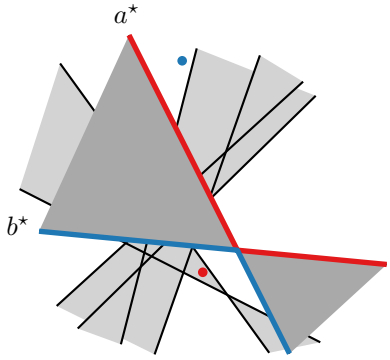
For a half-plane γ from the intersection (3), let ℓ_γ be its underlying line in the primal space. If γ is an upper half-plane, color the point in the dual space corresponding to ℓ_γ blue or red otherwise; see Figure 3. Then each upper half-plane γ corresponds, in the dual space, to a blue point that has at least $n - k$ lower v-shaped rays below (because the upper half-plane γ contains at least $n - k$ segment pieces). Similarly, if γ is a lower half-plane, it corresponds, in the dual space, to a red point that has at least $n - k$ upper v-shaped rays above; see Figure 3b. Alternatively, each blue point has less than k lower v-shaped rays above it and each red point has less than k upper v-shaped rays below it.

For any blue point q, let its *level* $\ell_\Lambda(q)$ be the number of lower v-shaped rays above q. Similarly, for any red point q, let its *level* $\ell_V(q)$ be the number upper v-shaped rays below q. Let $U_k(V)$ be the set of all blue points of level $\ell_V(q) < k$ and $L_k(\Lambda)$ be the set of all red points of level $\ell_\Lambda(q) < k$. Then, the linear program in the dual space becomes the following: given the set of blue and red points $K = L_k(\Lambda) \cup U_k(V)$ in the plane, compute

$$\begin{aligned} w(K) = \quad & \min\ \phi(h) \qquad (4) \\ & \text{s.t. } h \text{ is a line of the cone formed by } a_s^\star \text{ and } b_s^\star \\ & \quad \text{all red points of } K \text{ are below } h \\ & \quad \text{all blue points of } K \text{ are above } h, \end{aligned}$$



(a) the segments S, an upper half-plane, and a lower half-plane



(b) dual image of S and the underlying lines of the two half-planes; the wedge $a^\star b^\star$ is dark grey

Figure 3: The segments S and their dual transformation. In (a) the segment ab intersects the lower half-plane (red) iff in (b) the red point is below the upper v-shaped ray (red). Similarly in (a) the segment ab intersects the upper half-plane (blue) iff in (b) the blue point is above the lower v-shaped ray (blue).

where $a_s^\star b_s^\star$ is the wedge in the dual space of the query segment s and $\phi(h)$ can be any linear function over the coefficients of h's half-plane equation.

Even though $L_k(\Lambda) \cup U_k(V)$ may contain infinitely many points, to construct our linear program (4) we only need to consider the points on the boundary of $L_k(\Lambda) \cup U_k(V)$. This boundary is the k-level in the arrangement of v-shaped rays V and Λ. To solve our problem, we could compute this region using standard algorithms for computing the k-level of a line arrangement [3] (this would give us the set of constraints for our linear program) and then solve the linear program. The size of this k-level may be superlinear, however, and the best upper bound so far is $O(nk^{1/3})$ [7].

We show how to design an efficient algorithm using Lemma 3. To divide our problem into subproblems we use the following geometric result.

Lemma 4 ((Chazelle 1993) Cutting Lemma [5]) *Given n lines and some $r \in \mathbf{N}$, we can cut the plane into $O(r^2)$ simplices (triangles) such that each simplex intersects at most $\lceil \alpha n \rceil$ lines, $\alpha = 1/r$. The construction takes linear time for constant r.*

For our input, n segments correspond to n pairs of lines; we can use Lemma 4 to cut the plane into triangles intersecting at most $\lceil \alpha n \rceil$ v-shaped rays by applying Lemma 4 for $2n$ lines.

5.1 Deciding whether $\overline{\dot{\mathrm{TD}}}(q, S) \geq k$

First we prove that we can decide whether $\overline{\dot{\mathrm{TD}}}(q, S) \geq k$ for some $q \in s$ in $O(n \log n)$ expected time.

Theorem 5 *For a segment s and an n-segment set S in the plane, we can decide whether $\overline{\dot{\mathrm{TD}}}(q, S) \geq k$ for some $q \in s$ in $O(n \log n)$ expected time.*

Proof. We use Lemma 3. To partition the problem into subproblems (see requirement 2 in Lemma 3) we will apply the cutting Lemma 4. To enable this application we need to solve a slight generalization of the dual problem: given a simplex Δ that intersects the set of v-shaped rays $W_\Delta \subseteq W$, $W_\Delta = V_\Delta \cup \Lambda_\Delta$ and has strictly x lower v-shaped rays below it and y upper v-shaped rays above it, compute $w(f(W_\Delta, \Delta, x, y))$, where

$$f(W_\Delta, \Delta, x, y) = (L_{k-x}(\Lambda) \cup U_{k-y}(V))$$

Let us now check that the requirements 1 and 2 of Lemma 3 can be fulfilled:

Requirement 1. The line h that optimizes w is defined by two points: one point is the intersection of $a_s^\star$ and $b_s^\star$, and the other is in $(L_{k-x}(\Lambda_\Delta) \cup U_{k-y}(V_\Delta))$. Therefore, for any subset of constraints $F \subseteq f(W_\Delta, \Delta, x, y)$, the basis $B \subseteq F$ is just one point $b \in F$, that is, $B = \{b\}$. Thus, to check whether basis $\{b\}$ satisfies $f(W_\Delta, \Delta, x, y)$

we only need to check whether for the line h defined by b and the intersection of $a_s^\star$ and $b_s^\star$ such that h is of the cone[1] formed by $a^\star$ and $b^\star$ the following holds: (a) $L_{k-x}(\Lambda_\Delta)\cap\Delta$ is completely below h and (b) $U_{k-y}(V_\Delta)\cap\Delta$ above h. We show how to verify (a); an analogous argument can be applied for (b).

(a) We need to test whether h crosses the region $L_{k-x}(\Lambda_\Delta)\cap\Delta$. Recall that the region $L_{k-x}(\Lambda_\Delta)$ is a $(k-x)$-level of n lower v-shaped rays, constructing this region might be expensive. However, for testing whether a line intersects the region we only need to solve the 1-dimensional problem: construct the 1-dimensional $(k-x)$-level of the intersection points of the lower v-shaped rays V_Δ on the line h and check if it intersects with the part of h contained in the triangle. This can be done in time $O(n\log n)$ by sorting the intersection points of the v-shaped rays Λ_Δ with the line h. Therefore, the entire basis satisfaction test can be done in time $D(n)=O(n\log n)$ time.

Requirement 2. For the initial input (W_Δ,Δ,x,y) of size n, we form the r triangles using the cutting Lemma 4. We then intersect these triangles with Δ and retriangulate Δ to obtain a triangulation of Δ for which the bound from Lemma 4 clearly holds. We then discard those triangles that do not intersect the wedge formed by $a_s^\star$ and $b_s^\star$ (representing the segment in the primal), since those triangles do not have a feasible solution. Let the set of the resulting triangles be $\Delta_1,\ldots,\Delta_{r'}$ for $r'\le r$. Then,

$$f(W_\Delta,\Delta,x,y)=\bigcup_{i=1}^{r'} f(W_i,\Delta_i,x+x_i,y+y_i),$$

where W_i denotes the set of v-shaped rays of W_Δ intersecting Δ_i (of size at most $\lceil\alpha n\rceil$), and x_i and y_i denote the number of v-shaped rays strictly below Δ_i and above Δ_i.

The theorem follows from Lemma 3. □

5.2 Computing $\overline{\mathrm{TD}}(s,S)$

To find $\overline{\mathrm{TD}}(s,S)$ for the segment s and the set S of n segments, we consider a more general linear programming problem, where each constraint has a label k and the objective is to maximize k such that there exists a point inside all given half-planes with label less than k, or in dual form

$$w'(K)=\min\ (-k,\phi(h)) \qquad (5)$$
$$\text{s.t. all red points of } K \text{ with label } <k \text{ are below } h$$
$$\text{all blue points of } K \text{ with label } <k \text{ are above } h,$$

where h is a line of the cone formed by $a_s^\star$ and $b_s^\star$, and the minimum is taken lexicographically. For example, a special case of the above is the following problem: given a sequence C of linear constraints, find the longest prefix of C whose linear program is feasible. Interestingly, as observed by Chan [4], this problem is LP-type and can be solved in $O(|S|)$ expected time; Chan also showed that w' in (5) is LP-type of constant dimension.

The problem of computing $\overline{\mathrm{TD}}(s,S)$ dualizes to finding $w'(f'(W_\Delta,\Delta,x,y))$ for $\Delta=\mathbb{R}^2$ and x = y = 0, where

$$f'(W_\Delta,\Delta,x,y)=\{q\in\Delta \text{ in red with label } \ell_\Lambda(q)+x\}$$
$$\cup\ \{q\in\Delta \text{ in blue with label } \ell_V(q)+y\}.$$

Theorem 6 *For a segment s and a set S of n segments in the plane, we can compute $\overline{\mathrm{TD}}(s,S)$ in $O(n\log n)$ expected time.*

Proof. We have to show again that the requirements 1 and 2 of Lemma 3 hold.

For the requirement 1 we need to test whether a basis satisfies $f'(W_\Delta,\Delta,x,y)$ with respect to w', which again reduces to testing whether $L_{k-x}(\Lambda_\Delta)\cap\Delta$ is below h and $U_{k-y}(V_\Delta)\cap\Delta$ is above h. As in the proof of Theorem 5, this requires $D(n)=O(n\log n)$ time.

For the requirement 2, we can form the simplices Δ_i and indices x_i, y_i for $i=1,\ldots,r'$ similarly as in the proof of Theorem 5. Therefore, we have

$$f'(W_\Delta,\Delta,x,y)=\bigcup_{i=1}^{r'} f(W_i,\Delta_i,x+x_i,y+y_i).$$

The theorem follows from Lemma 3. □

6 Discussion

We have generalized three classical combinatorial depth measures for sets of points in the plane to sets of segments. For a given set S of n line segments and a given segment s in $\mathbb{R}^2$, we compute our generalization of Tukey depth $\overline{\mathrm{TD}}(s,S)$ in optimal time, as our algorithm takes the same expected time $O(n\log n)$ as for computing the Tukey depth $\mathrm{TD}(q,P)$ for a point q relative to a set of n points in $\mathbb{R}^2$ (for which a lower bound of $\Omega(n\log n)$ time is known [2]). On the other hand, we need $O(n^2)$ time and expected $O(n^{4/3}\log n)$ time, respectively, to compute our generalization of simplicial depth $\overline{\mathrm{SD}}(s,S)$ and eutomic depth $\overline{\mathrm{ED}}(s,S)$, whereas both $\mathrm{SD}(q,P)$ and $\mathrm{ED}(q,P)$ can be computed in time $O(n\log n)$ for sets of points. It remains open to determine whether $\overline{\mathrm{SD}}(s,S)$ and $\overline{\mathrm{ED}}(s,S)$ can be computed faster (a lower bound of $\Omega(n\log n)$ time for computing simplicial depth is also known [2]).

References

[1] Greg Aloupis. Geometric measures of data depth. In *DIMACS Series in Discrete Mathematics and*

[1] Recall that in the primal this enforces the point to be on the segment.

Theoretical Computer Science, volume 72, pages 147–158, 2006.

[2] Greg Aloupis, Carmen Cortés, Francisco Gómez, Michael Soss, and Godfried Toussaint. Lower bounds for computing statistical depth. *Computational Statistics & Data Analysis*, 40(2):223–229, 2002.

[3] Timothy M. Chan. Remarks on k-level algorithms in the plane. Manuscript, 1999.

[4] Timothy M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 430–436, 2004.

[5] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993.

[6] Pierre Lafaye de Micheaux, Pavlo Mozharovskyi, and Myriam Vimond. Depth for curve data and applications. *Journal of the American Statistical Association*, 116(536):1881–1897, 2021.

[7] Tamal K. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19:373–382, 1998.

[8] Stephane Durocher, Robert Fraser, Alexandre Leblanc, Jason Morrison, and Matthew Skala. On combinatorial depth measures. *International Journal of Computational Geometry & Applications*, 28(4):381–398, 2018.

[9] Stephane Durocher and Spencer Szabados. Curve stabbing depth: Data depth for plane curves. In *Proc. 34th Canadian Conference on Computational Geometry (CCCG)*, pages 121–128, 2022.

[10] Herbert Edelsbrunner. *A Short Course in Computational Geometry and Topology*. SpringerBriefs in Applied Sciences and Technology. Springer International Publishing, 2014.

[11] Regina Y. Liu. On a notion of data depth based on random simplices. *Annals of Statistics*, 18(1):405–414, 1990.

[12] Peter J. Rousseeuw and Ida Ruts. Bivariate location depth. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 45(4):516–526, 12 1996.

[13] John W. Tukey. Mathematics and the picturing of data. *International Congress of Mathematicians*, 2:523–531, 1975.

How Small Can Faithful Sets Be? Ordering Topological Descriptors

Brittany Terese Fasy * David L. Millman † Anna Schenfisch ‡

Abstract

Recent developments in shape reconstruction and comparison call for the use of many different (topological) descriptor types, such as persistence diagrams and Euler characteristic functions. We establish a framework to quantitatively compare the strength of different descriptor types, setting up a theory that allows for future comparisons and analysis of descriptor types and that can inform choices made in applications. We use this framework to partially order a set of six common descriptor types. We then give lower bounds on the size of sets of descriptors that uniquely correspond to simplicial complexes, giving insight into the advantages of using verbose rather than concise topological descriptors.

1 Introduction

The persistent homology transform and Euler characteristic transform were first explored in [41], which shows the uncountable set of persistence diagrams (or Euler characteristic functions, respectively) corresponding to lower-star filtrations in every possible direction uniquely represents the shape being filtered. That is, the uncountable set of topological descriptors is *faithful* for the shape. Faithfulness of topological transforms is closely related to *tomography* [22, 39], and the alternate proof of faithfulness given in [16] makes use of tools from this field. Of course, applications can only use finite sets of descriptors, which are not guaranteed to be faithful. This motivates theoretical work on finding finite faithful sets of descriptors [3, 7, 13, 32], and such work supports the use of topological descriptors in shape comparison applications. Many descriptor types are used in applications, such as versions of persistence diagrams [4, 21, 23, 37, 42, 45], Euler characteristic functions [1, 6, 20, 26, 31, 33, 36], Betti functions [15, 25, 35, 38, 44, 46], and others [2, 17, 40].

Faithfully representing a shape with a small number of descriptors is desirable for computational and storage reasons. How, then, should investigators choose the particular topological descriptor type to use in applications? While computational complexities of computing each topological descriptor type are well-studied, it is not yet known how the use of particular descriptor types impacts the minimum size of faithful sets. This uncertainty motivates our main questions: *how can we rigorously compare descriptor types in terms of their ability to uniquely correspond to shapes*, and *how do popular descriptor types compare?*

We prove the parital order of Figure 1. Additionally, we provide lower bounds on the cardinality of faithful sets for both concise and verbose descriptors, and identify properties that indicate concise descriptors are generally much weaker than verbose descriptors. This suggests applications research may benefit from the use of verbose descriptors instead of the more widely adopted concise descriptors.

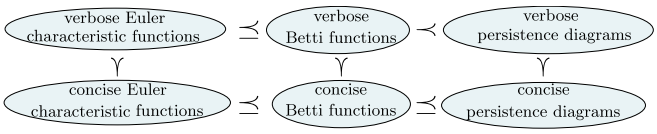


Figure 1: Summary of the relations between common descriptor types. For example, concise persistence diagrams are always at least as efficient as concise Betti functions at forming faithful sets, where efficiency is measured by cardinality of faithful sets.

2 Preliminary Considerations

In this section, we provide background and definitions used throughout.

Simplicial Complexes and Filtrations We assume the reader is familiar with foundational ideas from topology, such as homology, Betti number (β_k) and Euler characteristic (χ). See, e.g., [9, 18]. For a simplicial complex K and $i \in \mathbb{N}$, we use the notation K_i for the set of its i-simplices and n_i as the number of i-simplices. Furthermore, we assume our simplicial complexes are *abstract* simplicial complexes immersed in Euclidean space such that each simplex is embedded and the vertices are in general position; see Assumption 1 in Appendix C. A *filter* of K is a map $f\colon K \to \mathbb{R}$ such that, each sublevel set $f^{-1}(-\infty, t]$ is either empty or a simplicial complex. Letting $F(t) := f^{-1}(-\infty, t]$, the sequence $\{F(t)\}_{t\in\mathbb{R}}$ is the *filtration* associated to f. For each $k \in \mathbb{N}$, the inclusion $F(i) \hookrightarrow F(j)$ induces a linear map on homology, $H_k(F(i)) \to H_k(F(j))$. We write $\beta_k^{i,j}(K, f)$ to mean rank of this map, or simply $\beta_k^{i,j}$ if K and f are clear from context. We call a filter function $f' : K \to \{1, 2, \ldots, \#K\}$ a *compatible index filter* for f if, for all $\tau, \sigma \in K$ with $f(\tau) \le f(\sigma)$, then $f'(\tau) \le f'(\sigma)$. Every filter function has at least one compatible index filter.

The *lower-star filter* of a simplicial complex K immersed in $\mathbb{R}^d$ with respect to some direction $s \in \mathbb{S}^{d-1}$,

*School of Computing and Dept. Mathematical Sciences, Montana State U., brittany.fasy@montana.edu

†Blocky Inc., david@blocky.rocks

‡Dept. Mathematics and Computer Science, Eindhoven U. of Technology, a.k.schenfisch@tue.nl

is the map $f_s : K \to \mathbb{R}$ that takes a simplex σ to the maximum height of its vertices with respect to direction s, i.e., $f_s(\sigma) := \max\{s \cdot v \mid v \in K_0 \cap \sigma\}$, where $s \cdot v$ denotes the dot product.

Faithfully Representing a Simplicial Complex Since we define relations based on the ability of descriptor types to represent particular filtrations of simplicial complexes, we take the following definition.

Definition 1 (Topological Descriptors) *A* (topological) descriptor type *is a map whose domain is the collection of filtered simplicial complexes. Given such map,* D*, a* (topological) descriptor of type *D is the image of a specific filtered simplicial complex under D.*

When considering many filtrations of the same simplicial complex, we may index the filtrations by some parameter set, P. If a descriptor of type D corresponds to a filtration of a simplicial complex K where the filtration is parameterized by $p \in P$, we use the notation $\mathrm{D}(K, p)$, or $\mathrm{D}(p)$ when K is clear from context. We refer to the parameterized set of descriptors as $\mathrm{D}(K, P) := \{(p, \mathrm{D}(K, p))\}_{p \in P}$.

We compare descriptor types by quantifying the number of descriptors needed to uniquely identify a shape. Specifically:

Definition 2 (Faithful) *Let K be a simplicial complex, P parameterize a set of filtrations of K, and* D *be a topological descriptor type. We say that* $\mathrm{D}(K, P)$ is faithful *if, for any simplicial complex L we have the equality* $\mathrm{D}(L, P) = \mathrm{D}(K, P)$ *if and only if $L = K$.*

We note here that $|\mathrm{D}(K, P)| = |P|$. See Appendix B for a reformulation of Definition 2 in terms of set intersections, as well as a lemma that can be gained from this perspective.

3 Six Common Descriptor Types

The set we partially order is the strength equivalence classes of six popular descriptor types, which we define here. We begin with concise persistence diagrams.

Definition 3 (Concise Persistence Diagram) *Let $f : K \to \mathbb{R}$ be a filter function. For $k \in \mathbb{N}$, the* k-dimensional persistence diagram *is:*

$$\rho_k^f := \big\{(i,j)^{\mu^{(i,j)}} \ \text{s.t.}\ (i,j) \in \overline{\mathbb{R}}^2 \\ \text{and } \mu^{(i,j)} = \beta_k^{i,j-1} - \beta_k^{i,j} - \beta_k^{i-1,j-1} + \beta_k^{i-1,j}\big\},$$

where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$ and $(i,j)^m$ denotes m copies of the point (i,j). The concise persistence diagram *of f, denoted ρ^f, is the indexed union of all k-dimensional concise persistence diagrams $\rho^f := \cup_{k \in \mathbb{N}} \rho_k^f$.*

Since simplices can appear at the same parameter value in a filtration, not all cycles are represented in the persistence diagram. However, having every simplex "appear" in a topological descriptor is helpful, in addition to being natural. Thus, we introduce *verbose* descriptors, which contain this information. We define verbose descriptors via compatible index filtrations; by Lemma 52 and Corollaries 54-55 of [12], the resulting descriptors are well-defined and independent of our choice of compatible index filtration. We begin with *verbose persistence diagrams*:

Definition 4 (Verbose Persistence Diagram) *Let $f : K \to \mathbb{R}$ be a filter for K, and let f' be a compatible index filter. For $k \in \mathbb{N}$, the* k-dimensional verbose persistence diagram *is the following multiset:*

$$\tilde{\rho}_k^f := \left\{(f(\sigma_i), f(\sigma_j)) \ \text{ s.t. } (i,j) \in \rho_k^{f'}\right\}.$$

The verbose persistence diagram *of f, denoted $\tilde{\rho}^f$, is the indexed union of all $\tilde{\rho}_k^f$.*

Recording invariants other than homology leads to other topological descriptor types; recording Betti numbers gives us *Betti functions*.

Definition 5 (Betti Functions) *Let $f\colon K \to \mathbb{R}$ be a filter function. The* kth Betti function, *$\beta_k^f : \mathbb{R} \to \mathbb{Z}$, is defined by*

$$\beta_k^f(t) := \beta_k\left(f^{-1}(-\infty, t]\right).$$

The indexed collection of Betti functions for all dimensions, $\beta^f := \{\beta_k^f \mid k \in \mathbb{N}\}$, is the Betti function.

Let f' be an index filter compatible with filter function f. We call $\sigma \in K$ positive *(respectively,* negative*)* for *β_k if the inclusion of σ into the index filtration of f' increases (resp., decreases) β_k. We denote the positive (resp., negative) simplices by $K_k^+ \subseteq K_k$ (and $K_{k+1}^- \subseteq K_{k+1}$). Then, the* kth verbose Betti function, *$\tilde{\beta}_k^f : \mathbb{R} \to \mathbb{Z}^2$, is defined by*

$$\tilde{\beta}_k^f(p) := \Big(\left|\{\sigma \in K_k^+ \ \text{s.t. } f(\sigma) \le p\}\right|, \\ \left|\{\sigma \in K_{k+1}^- \ \text{s.t. } f(\sigma) \le p\}\right|\Big).$$

The collection of verbose Betti number functions for each dimension is known as the verbose Betti function *and is denoted $\tilde{\beta}^f$.*

If we record *Euler characteristic* in a filtration, we obtain *Euler characteristic functions.*[1]

Definition 6 (Euler Characteristic Functions) *Let $f\colon K \to \mathbb{R}$ be a filter function. The* Euler characteristic function, *$\chi^f : \mathbb{R} \to \mathbb{Z}$, is defined by:*

$$\chi^f(p) := \chi\left(\{f^{-1}(-\infty, p]\}\right).$$

Let f' be an index filter compatible with f. We call $\sigma \in K$ even *(respectively,* odd*) if the dimension of σ is even (resp., odd). Denoting the set of even (resp., odd) simplices by E (and O), the* verbose Euler characteristic function, *$\widetilde{\chi}^f : \mathbb{R} \to \mathbb{Z}^2$, is defined by*

$$\widetilde{\chi}^f(p) := \left(|\sigma \in E \ \text{s.t. } f(\sigma) \le p|, |\sigma \in O \ \text{s.t. } f(\sigma) \le p|\right).$$

In other words, $\tilde{\chi}^f$ represents χ^f as a parameterized count of even- and odd-dimensional simplices.

[1] Euler characteristic functions and Betti functions are sometimes called Euler (characteristic) curves or Betti curves.

In each of the descriptor types above, we drop the superscript f when it is clear from context. See Appendix A for examples of these descriptor types. While concise descriptors may feel more familiar, verbose descriptors are not new. Many algorithms for computing persistence (e.g., [9, Chapter VII]), explicitly compute events with trivial lifespan. In [27], the definition of persistence diagrams agrees with our Definition 4. Verbose descriptors are closely connected to the charge-preserving morphisms of [14, 28]. In [43], verbose persistence is defined via filtered chain complexes; [5, 29, 30, 47] also take this view as a foundational definition. The behavior of verbose versus concise descriptors is explored in [10, 30, 48].

Verbose (concise) descriptors are sometimes called augmented (non-augmented, respectively) in the literature. We refer to points on a verbose diagram with zero-lifespan as *instantaneous*. Such points correspond to length-zero barcodes in a verbose barcode, which are sometimes referred to as *ephemeral*.

While we chose the six descriptor types above due to their relevance in applications, we emphasize that Definition 1 is very general. We explore a few pathological descriptor types in Appendix C.

4 Relating Descriptor Types

We now develop tools to compare descriptor types, by comparing the sizes of faithful sets. Given a topological descriptor type D and a simplicial complex K immersed in $\mathbb{R}^d$, let the infimum size of faithful sets for K be denoted

$$\Gamma(K, D) := \inf_{D(K,P) \text{ faithful}} \{|P|\}.$$

Intuitively, the stronger D is, the smaller $\Gamma(K, D)$. Often, we find $\Gamma(K, D)$ is finite. For some descriptors and K, we find $\Gamma(K, D) = \aleph_0$ (the cardinality of $\mathbb{N}$) or $\Gamma(K, D) = \aleph_1$ (the cardinality of $\mathbb{R}$); see Appendix C for examples. If no faithful set of type D exists for K, we write $\inf_{x \in \emptyset}\{x\} = \aleph_\top$, and we think of this as "the highest" cardinality. By the axiom of choice, $\aleph_0 < \aleph_1$; see e.g., [19, Ch. 2]. Thus, we have a total order on possible values of $\Gamma(K, D)$:

$$c < \aleph_0 < \aleph_1 < \aleph_\top,$$

where $c \in \mathbb{N}$.

Definition 7 (Strength Relation) *Let A and B be two topological descriptor types. If, for every simplicial complex K immersed in $\mathbb{R}^d$, we have $\Gamma(K, A) \geq \Gamma(K, B)$, then we say that A is* weaker *than B (and B is* stronger *than A) denoted $[A] \preceq [B]$. If $[A] \preceq [B]$ and $[B] \preceq [A]$, then we say that A and B have equal strength, denoted $[A] = [B]$.*

The relations $=$ are $\preceq$ are well-defined on strength equivalence classes. See Lemma 16 in Appendix D.1. Also see Example 1 of Appendix C for two different descriptor types in the same equivalence class.

We write $[A] \prec [B]$ if $[A] \preceq [B]$ and $[A] \neq [B]$. That is, if $[A] \preceq [B]$ and there exists a simplicial complex for which the minimum faithful set of type B is strictly smaller than that of type A, or for which there exists a faithful set of type B but not of type A. Descriptor types need not be comparable; see Lemma 14 of Appendix C.

We conclude this section by defining reduction of one descriptor to another, and show this is a valid strategy for determining equivalence class order.

Definition 8 (Reduction) *Let A and B be two topological descriptor types. We say B is* reducible *to A if, for all simplicial complexes K and any filtration f of K, we can compute $A(f)$ from $B(f)$ alone.*[2]

Intuitively, B is at least as informative as A. More formally, we have the following lemma, whose proof is in Appendix D.1:

Lemma 1 *Let A and B be two topological descriptor types. If B is reducible to A, we have $[A] \preceq [B]$.*

5 A Proof of Partial Order

In this section, we provide a partial order on the six topological descriptors of Section 3. Omitted proofs are in Appendix D.2. While the results and definitions of previous sections were general, we now focus on descriptors corresponding to lower-star filtrations.

By simple reduction arguments, we immediately have the following lemma.

Lemma 2 *$[\chi] \preceq [\beta] \preceq [\rho]$ and $[\tilde{\chi}] \preceq [\tilde{\beta}] \preceq [\tilde{\rho}]$.*

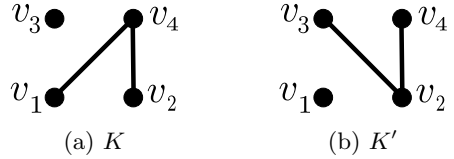
We also use reduction to order a class of a concise descriptor type and its verbose counterpart.

Lemma 3 *$[\chi] \preceq [\tilde{\chi}]$, $[\beta] \preceq [\tilde{\beta}]$, and $[\rho] \preceq [\tilde{\rho}]$.*

Next, we see that no concise class is equal to a verbose class.

Lemma 4 *For $\mathrm{D} \in \{\chi, \beta, \rho\}$ and $\tilde{\mathrm{D}} \in \{\tilde{\chi}, \tilde{\beta}, \tilde{\rho}\}$, we have $[\mathrm{D}] \neq [\tilde{\mathrm{D}}]$.*

The proof is given by considering a single edge in $\mathbb{R}^2$, and showing a minimal faithful set of type $\tilde{\mathrm{D}}$ has cardinality two; whereas, a minimal faithful set of type D has cardinality at least three; see Appendix D.2 for full details.



(a) K (b) K'

Figure 2: Complexes used in the proof of Lemma 5.

[2] In this reduction, we asusme the real-RAM model of computation.

In [13], faithfulness is shown via knowing the dimension of event rather than any birth-death pairings, so the resulting faithful sets of verbose persistence diagrams and verbose Betti functions have equal cardinality for every simplicial complex. One might wonder, then, if $[\tilde{\rho}]$ equals $[\tilde{\beta}]$. However, this is an incorrect leap; faithful sets of [13] are almost certainly not minimal. The next lemma gives an instance where birth-death pairings matter, rather than just an event's existence or dimension.

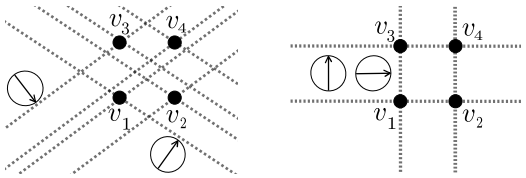
Lemma 5 $[\tilde{\beta}] \prec [\tilde{\rho}]$.

Proof. We know by Lemma 2 that $[\tilde{\beta}] \leq [\tilde{\rho}]$. We must show that equality does not hold; that is, that there exists a simplicial complex for which the cardinality of the minimal faithful sets of augmented Betti functions and augmented persistence diagrams differ. Consider the simplicial complex K in $\mathbb{R}^2$ consisting of: four vertices $v_1 = (0,0), v_2 = (0,1), v_3 = (1,0)$, and $v_4 = (1,1)$ and two edges $[v_1, v_4]$ and $[v_2, v_4]$, as in Figure 2(a).

Let $e_1 = (1,0)$ and $e_2 = (0,1)$. We first claim that $\tilde{\rho}(K, \{e_1, e_2\})$ is faithful, meaning $\Gamma(K, \tilde{\rho}) \leq 2$. Such diagrams uniquely identify the vertex set of K by [3, Lemma 4]; we provide further details here. From $\tilde{\rho}(e_1)$, we know K has four vertices, two with x-coordinate 0, and two with x-coordinate 1. Similarly, from $\tilde{\rho}(e_2)$, we know two of the four vertices have y-coordinate 0 and two have y-coordinate 1. There are exactly four ways to pair our x- and y-coordinates, so we know the locations of each vertex. See Figure 3(b).

For the edges, we first note that from $\tilde{\rho}(e_2)$ in degree zero we see an instantaneous birth/death at height one as well as a connected component born at height zero that dies at height one, so we know K has exactly two edges with height one in direction e_2. Namely, we know we have either the edges $[v_1, v_3]$ and $[v_2, v_3]$, or $[v_1, v_4]$ and $[v_2, v_4]$, i.e., we have one of the two complexes shown in Figure 2. Because $\tilde{\rho}(e_1)$ sees two zero-dimensional births at height zero with an infinite lifespan, we know there is no edge from v_1 to v_3. Finally, since higher homology is trivial, we know there are no other simplices and have determined K exactly; thus, we have a faithful set of size two.

We next show that $\Gamma(K, \tilde{\beta}) > 2$. Suppose, by way of contradiction, that s_1 and s_2 are two directions such that $\tilde{\beta}(K, \{s_1, s_2\})$ is a faithful set. We first show, without loss of generality, $s_1 \in \{e_1, -e_1\}$ and $s_2 \in \{e_2, -e_2\}$. Suppose this is not the case. Because $s_1 = -s_2$ does not correspond to a faithful set, we assume (wlog) that $s_1 \neq -s_2$. Then, at least one of s_1 or s_2 sees the vertices of K at more than two distinct heights; see Figure 3(a). In order to know the precise coordinates of each vertex, we need to correctly pair heights in directions s_1 and s_2. However, since at least one of s_1 or s_2 reports more than two distinct heights, we have more than four possible pairings (see also [3, Lemma 4]). We claim it is not possible to find the four correct pairings. The degree-zero information $\tilde{\beta}_0(s_1)$ and $\tilde{\beta}_0(s_2)$ alone is insufficient, as it only tells us the heights of vertices. From $\tilde{\beta}_1$, we know the height of edges, which only confirms the height of the top vertex and that there is some vertex below, information we already had from $\tilde{\beta}_0$. Thus, we must have $s_1 \in \{e_1, -e_1\}$ and $s_2 \in \{e_2, -e_2\}$. However, for each of these four directions, the associated verbose Betti function is not able to distinguish the two complexes shown in Figure 2(b). For instance, both $\tilde{\beta}(K, e_1)$ and $\tilde{\beta}(K', e_1)$ see two vertices at height zero, and two vertices and two edges at height one, i.e., $\tilde{\beta}(K, e_1) = \tilde{\beta}(K', e_1)$. The other cases of s_1 and s_2 are similar. Thus, we have found a faithful set of verbose persistence diagrams with cardinality two, but have shown any faithful set of verbose Betti functions must have cardinality greater than two. □



(a) (b)

Figure 3: For the given vertex set, heights of filtration events in the indicated directions are shown as dashed grey lines. While we know the number of vertices on each line, for two directions not both in $\{\pm e_1, \pm e_2\}$, as in (a), we can not identify vertex locations. Only when choosing one each of $\pm e_1$ and $\pm e_2$, as in (b), is the set of vertices satisfying these constraints unique.

Combining results, we arrive at our main theorem.

Theorem 6 (Partial Ordering) *The partial order of strength classes of topological descriptor types shown in Figure 1 is correct.*

6 Bounds on Faithful Sets

Here, we provide lower bounds on the size of faithful sets of the six descriptor types of Section 3.

6.1 Concise Descriptor Bounds

A defining feature of concise descriptors is that there are not generally events at every vertex height in a filtration. The closer a feature is to coplanar, the smaller the range of directions that can detect it becomes ([11, Sec. 4] explores this specifically for Euler characteristic functions). Difficulty detecting the presence or absence of structures near to the same affine subspace puts greater restrictions on the ability of concise descriptors to form faithful sets. We use the following definition to help this claim precise.

Definition 9 (Simplex Envelope) *Let K be a simplicial complex in $\mathbb{R}^d$, let $\sigma \in K$, and let $S \subseteq \mathbb{S}^{d-1}$. Then, we define the* envelope of σ, *denoted $\mathcal{E}_\sigma^S$, as the intersection of (closed) supporting halfspaces*

$$\mathcal{E}_\sigma^S = \bigcap_{s \in S} \{p \in \mathbb{R}^d \mid s \cdot p \geq \min_{v \in \sigma}(s \cdot v)\}.$$

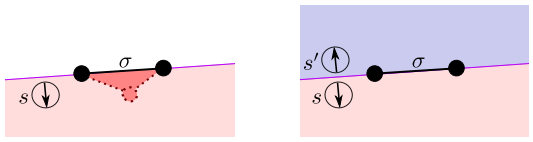
If S is clear from context, we write $\mathcal{E}_\sigma$. By the dimension *of $\mathcal{E}_\sigma$, we mean the largest dimension of ball that can be contained entirely in $\mathcal{E}_\sigma$.*

Since $\mathcal{E}_\sigma^S$ is an intersection of convex regions, it is itself convex. Furthermore, with respect to each $s \in S$, the height of each point of σ is greater than or equal to its minimum vertex, so $\mathcal{E}_\sigma^S$ contains σ.

Remark 1 *The simplex envelopes of Definition 9 have connections to well-studied topics such as convex cones, support functions, etc. See [8, 34]. In particular, [34, Thm 3.1.1, Cor 3.1.2] establish that a simplex envelope corresponding to the entire sphere of directions is the simplex itself.*

We use simplex envelopes to define a necessary condition for concise descriptors to form a faithful set.

Lemma 7 (Envelopes for Faithful Concise Sets) *Let K be a simplicial complex immersed in $\mathbb{R}^d$, let $D \in \{\chi, \beta, \rho\}$, and let $S \subseteq \mathbb{S}^{d-1}$ so that $\mathrm{D}(K, S)$ is faithful. Then, for any maximal simplex σ in K, the dimension of $\mathcal{E}_\sigma$ equals the dimension of σ.*



(a) (b)

Figure 4: With only the single direction s perpendicular to maximal edge σ in $\mathbb{R}^2$, the envelope $\mathcal{E}_\sigma^s$ is two-dimensional. Then, we could place an adversarial two-simplex contained in $\mathcal{E}_\sigma^s$ that is undetectable by $D(s)$, for $D \in \{\chi, \beta, \rho\}$, as in (a). In (b), the inclusion of s' reduces $\mathcal{E}_\sigma^{\{s,s'\}}$ to a linear subspace (purple intersection of pink and blue halfspaces) and the adversarial two-simplex would be detected by $D(s')$.

A proof of Lemma 7 appears in Appendix D.3. See Figure 4 for an example of what might go wrong if a simplex envelope does not satisfy the conditions of Lemma 7. Since we require the envelopes of a k-simplex to be k-dimensional, and since envelopes are the intersections of closed half spaces, standard arguments from manifold theory give us the following.

Corollary 1 (Concise Descriptors Per Maximal Simplex) *Let K be a simplicial complex immersed in $\mathbb{R}^d$, let $\mathrm{D} \in \{\chi, \beta, \rho\}$, and let $S \subseteq \mathbb{S}^{d-1}$. If $\mathrm{D}(K, S)$ is faithful, then for each maximal simplex $\sigma \in K$ of dimension $k < d$, the set S has at least $d - k + 1$ directions perpendicular to σ. If $k < d - 1$, these directions are pairwise linearly independent.*

Lemma 7 and Corollary 1 each give us the following.

Corollary 2 (Tight Lower Bound) *Let K be a simplicial complex in $\mathbb{R}^d$, $\mathrm{D} \in \{\chi, \beta, \rho\}$, and $S \subseteq \mathbb{S}^{d-1}$. Suppose that $\mathrm{D}(K, S)$ is faithful. Then, $|S| \geq d + 1$, and this bound is tight.*

This bound is met whenever K is a single vertex. However, minimal faithful sets of concise descriptors are generally much larger. Counteracting the need for perpendicular directions is the fact that, as d increases, more simplices span common hyperplanes, so perpendicular directions can increasingly be shared. We use these observations to lower bound bound the worst-case size of faithful set of consice descriptors.

Theorem 8 (Lower Bound for Worst-Case Concise Descriptor Complexity) *Let $\mathrm{D} \in \{\chi, \beta, \rho\}$ and let K be a simplicial complex in $\mathbb{R}^d$ with n_1 edges. Then, the worst-case cardinality of a minimal descriptor set of type D is $\Omega(d + n_1)$.*

Proof. We construct a simplicial complex, K, and bound the minimum cardinality of a faithful set for K. Suppose that, for $d > 2$, that K is a graph in $\mathbb{R}^d$ with $n_1 < d - 1$ edges, and for some $S \subseteq \mathbb{S}^{d-1}$, the set $\mathrm{D}(K, S)$ is faithful. Then, by Lemma 7, the envelope of each maximal edge σ must be one-dimensional. Then, by Corollary 1, for every such σ, S contains $d - 1 + 1 = d$ pairwise linearly independent directions perpendicular to σ. Let S^* be a minimal subset of directions in S satisfying the conditions of perpendicularity and one-dimensional envelopes.

To build S^*, first note all edges of K are contained in a common n_1-plane, so there is a $(d - n_1 - 1)$-sphere's worth of directions perpendicular to *all* edges simultaneously. Such directions are maximally efficient in the sense that each can "count" for all edges at once. We choose any $d - 1$ pairwise linearly independent directions from this sphere to be included in S^*. Now we need an additional perpendicular direction for each edge to bring the total for each edge to d. To ensure the envelopes of each edge are one-dimensional, these additional directions must not be perpendicular to any hyperplane defined by subsets of more than one edge. This means we must consider a total of n_1 additional directions, so that S^* has cardinality $d - 1 + n_1$. Since $|S^*|$ lower bounds $|S|$, we find $|S| \in \Omega(d + n_1)$. □

6.2 Verbose Descriptor Bounds

We now shift to verbose descriptors, and begin with the tight lower bound.

Lemma 9 (Tight Lower Bound) *Let K be a simplicial complex in $\mathbb{R}^d$ and $\tilde{\mathrm{D}} \in \{\tilde{\chi}, \tilde{\beta}, \tilde{\rho}\}$. Suppose for some $S \subseteq \mathbb{S}^{d-1}$ the set $\tilde{\mathrm{D}}(K, S)$ is faithful. Then, $|S| \geq d$, and this bound is tight.*

Proof. No vertex in K can be described using fewer than d coordinates, so a set of descriptors of type $\tilde{\mathrm{D}}$ with cardinality less than d can never be faithful. To see that this bound is tight, when K is a single vertex, verbose descriptors generated by any d pairwise linearly independent directions form a faithful set. □

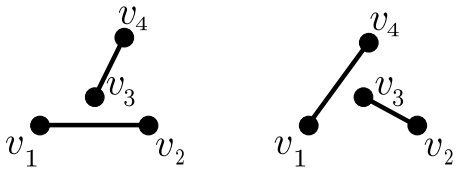
Next, we identify a family of simplicial complexes for which minimal faithful sets of verbose descriptors are at least linear in the number of vertices. We

use $\alpha_{i,j}$ to denote the angle that vector $v_j - v_i$ makes with the x-axis, taking value in $[0, 2\pi)$. We first observe a consequence of a specific instance of the general phenomenon that a simplicial complex stratifies the sphere of directions based on vertex order [7, 24].

Observation 1 *Suppose a simplicial complex in $\mathbb{R}^2$ contains an isolated edge $[v_1, v_2]$. Then, a birth event occurs height $s \cdot v_1$ in $\tilde{\rho}(K, s)$ for all s the interval $I = (\alpha_{1,2} - \pi/2, \alpha_{1,2} + \pi/2)$ of $\mathbb{S}^1$ (all s so that $s \cdot v_1 > s \cdot v_2$) and as an instantaneous event for all $s \in I^C$.*

We now construct the building block that forms the complexes used in our bound.

Construction 1 (Clothespin Motif) *Let K be a simplicial complex in $\mathbb{R}^2$ with a vertex set $\{v_1, v_2, v_3, v_4\}$ such that only v_3 is in the interior of the convex hull of $\{v_1, v_2, v_4\}$, and that the edge set consists of $[v_1, v_2]$ and $[v_3, v_4]$. See Figure 5a.*



(a) K (b) K'

Figure 5: The two simplicial complexes of Lemma 10.

Construction 1 was built specifically for the following necessary condition, the proof of which may be found in Appendix D.3.

Lemma 10 (Clothespin Representability) *Let K be as in Construction 1, and suppose that $\tilde{\rho}(K, S)$ is faithful. Then, there is some $s \in S$ so that the angle formed between s and the x-axis lies in the region*

$$W = [\alpha_{3,2} - \pi/2, \alpha_{3,4} - \pi/2] \cup [\alpha_{3,2} + \pi/2, \alpha_{3,4} + \pi/2].$$

We call W, the intervals of directions in $\mathbb{S}^1$ for which corresponding verbose descriptors can distinguish K from K' a clothespin's *region of observability* (similar to observability for concise Euler characteristic functions in [7, 11]). Crucially, W is defined by $\measuredangle v_2 v_3 v_4$, so we have the following.

Remark 2 (W Can be Arbitrarily Small) *As the angle $\measuredangle v_2 v_3 v_4$ approaches zero, the region of observability from Lemma 10 also approaches zero.*

We use Remark 2 to piece together clothespins so their regions of observability do not overlap.

Construction 2 (Clothespins on a Clothesline) *Let $K^{(m)}$ be a simplicial complex in $\mathbb{R}^2$ formed by m copies of Construction 1 (m clothespin motifs) such that the regions of observability for each clothespin do not intersect. This is possible for any m by Remark 2.*

See Figure 6. Construction 2 implies a lower bound on the worst-case size of faithful sets of verbose persistence diagrams.

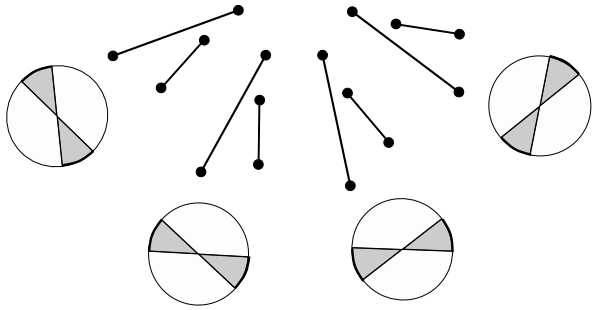


Figure 6: An example of $K^{(m)}$ for $m = 4$. Regions of observability are shown below each clothespin. By construction, each of these regions of $\mathbb{S}^1$ are disjoint.

Lemma 11 (Verbose Persistence Diagram Complexity) *Let $K^{(m)}$ be as in Construction 2 and suppose $\tilde{\rho}(K^{(m)}, S)$ is a faithful set. Then, S contains at least one direction in each of the m regions of observability, so $|S| \geq m = n_0/4$. Thus, $|S|$ is $\Omega(n_0)$.*

By Theorem 6, Lemma 11 implies the following:

Theorem 12 (Lower Bound for Worst-Case Verbose Descriptor Complexity) *Let $\tilde{\mathrm{D}} \in \{\tilde{\chi}, \tilde{\beta}, \tilde{\rho}\}$. Then, the worst-case cardinality of a minimal descriptor set of type $\tilde{\mathrm{D}}$ is $\Omega(n_0)$.*

7 Discussion

We provide a framework for comparing topological descriptor types by their ability to efficiently represent simplicial complexes. The tools developed here are a first step towards more theoretical justifications for the use of particular descriptor types in applications.

We focus on the descriptors that are particularly relevant to applications and related work, and give a partial order on this set of six descriptors, including the strict inequality, $[\tilde{\beta}] \prec [\tilde{\rho}]$. We also identify tight lower bounds for descriptor types, as well as bounds for worst-case complexity of sizes of faithful sets. Because faithful sets of concise descriptors require many perpendicular directions to each maximal simplex, a huge hiderance in practice, we believe applications research may benefit from the use of verbose descriptors rather than the current standard of concise descriptors. We are investigating other descriptor types in this framework as well, including merge trees.

Perhaps the strength classes $[\chi], [\beta]$, and $[\rho]$ intuitively feel as though they should be related by strict inequalities. However, this issue is nuanced. Lemma 15 (Appendix C) shows the impact that general position assumptions have on relations in this set. But even with general position, the seemingly advantageous "extra" information of homology compared to, e.g., Euler characteristic may no longer be so useful when we require tight envelopes around each maximal simplex. That is, once we have all the (many) required directions, we have already carved out the space filled by the complex, and already know quite a lot simply from the presence of events. Non-equality/equality of concise descriptors remains an area active of research.

References

[1] E. J. Amézquita, M. Y. Quigley, T. Ophelders, J. B. Landis, D. Koenig, E. Munch, and D. H. Chitwood. Measuring hidden phenotype: Quantifying the shape of barley seeds using the Euler characteristic transform. *in silico Plants*, 4(1):diab033, 2022.

[2] L. Batakci, A. Branson, B. Castillo, C. Todd, E. W. Chambers, and E. Munch. Comparing embedded graphs using average branching distance. *Involve, a Journal of Mathematics*, 16(3):365–388, 2023.

[3] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Reconstructing embedded graphs from persistence diagrams. *Computational Geometry: Theory and Applications*, 90, October 2020.

[4] P. Bendich, J. S. Marron, E. Miller, A. Pieloch, and S. Skwerer. Persistent homology analysis of brain artery trees. *The Annals of Applied Statistics*, 10(1):198, 2016.

[5] W. Chachólski, B. Giunti, A. Jin, and C. Landi. Decomposing filtered chain complexes: Geometry behind barcoding algorithms. *Computational Geometry*, 109:101938, 2023.

[6] L. Crawford, A. Monod, A. X. Chen, S. Mukherjee, and R. Rabadán. Predicting clinical outcomes in glioblastoma: an application of topological and functional data analysis. *Journal of the American Statistical Association*, 115(531):1139–1150, 2020.

[7] J. Curry, S. Mukherjee, and K. Turner. How many directions determine a shape and other sufficiency results for two topological transforms. *Transactions of the American Mathematical Society, Series B*, 9(32):1006–1043, 2022.

[8] J. Darrotto. Convex optimization and Euclidean distance geometry. *Palo Alto: Meboo*, 2013.

[9] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[10] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. Challenges in reconstructing shapes from Euler characteristic curves, 2018. arXiv:1811.11337.

[11] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. Challenges in reconstructing shapes from Euler characteristic curves. *Fall Workshop Computational Geometry*, 2018.

[12] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. A faithful discretization of the verbose persistent homology transform, 2019. arXiv:1912.12759.

[13] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. Efficient graph reconstruction and representation using augmented persistence diagrams. *Proceedings of the 34th Annual Canadian Conference on Computational Geometry*, 2022.

[14] B. T. Fasy and A. Patel. Persistent homology transform cosheaf, 2022.

[15] P. Frosini and C. Landi. Persistent Betti numbers for a noise tolerant shape-based approach to image retrieval. *Pattern Recognition Letters*, 34(8):863–872, 2013.

[16] R. Ghrist, R. Levanger, and H. Mai. Persistent homology and euler integral transforms. *Journal of Applied and Computational Topology*, 2:55–60, 2018.

[17] C. Giusti, E. Pastalkova, C. Curto, and V. Itskov. Clique topology reveals intrinsic geometric structure in neural correlations. *Proceedings of the National Academy of Sciences*, 112(44):13455–13460, 2015.

[18] A. Hatcher. Algebraic topology, Cambridge Univ. *Press, Cambridge*, 2002.

[19] T. Jech. Set theory. *Journal of Symbolic Logic*, pages 876–77, 1981.

[20] Q. Jiang, S. Kurtek, and T. Needham. The weighted Euler curve transform for shape and image analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 844–845, 2020.

[21] P. Lawson, J. Schupbach, B. T. Fasy, and J. W. Sheppard. Persistent homology for the automatic classification of prostate cancer aggressiveness in histopathology images. In *Medical Imaging 2019: Digital Pathology*, volume 10956, page 109560G. International Society for Optics and Photonics, 2019.

[22] V. Lebovici. Hybrid transforms of constructible functions. *Foundations of Computational Mathematics*, pages 1–47, 2022.

[23] Y. Lee, S. D. Barthel, P. Dłotko, S. M. Moosavi, K. Hess, and B. Smit. Quantifying similarity of pore-geometry in nanoporous materials. *Nature Communications*, 8:15396, 2017.

[24] J. Leygonie, S. Oudot, and U. Tillmann. A framework for differential calculus on persistence barcodes. *Foundations of Computational Mathematics*, pages 1–63, 2021.

[25] J. Li, L. Yang, Y. He, and O. Fukuda. Classification of hand movements based on EMG signals using topological features. *International Journal of Advanced Computer Science and Applications*, 14(4), 2023.

[26] L. Marsh and D. Beers. Stability and inference of the Euler characteristic transform, 2023. arXiv:2303.13200.

[27] A. McCleary and A. Patel. Edit distance and persistence diagrams over lattices. *SIAM Journal on Applied Algebra and Geometry*, 6(2):134–155, 2022.

[28] A. McCleary and A. Patel. Edit distance and persistence diagrams over lattices. *SIAM Journal on Applied Algebra and Geometry*, 6(2):134–155, 2022.

[29] F. Mémoli and L. Zhou. Stability of filtered chain complexes, 2022. arXiv:2208.11770.

[30] F. Mémoli and L. Zhou. Ephemeral persistence features and the stability of filtered chain complexes. In *39th International Symposium on Computational Geometry (SoCG 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[31] K. Meng, J. Wang, L. Crawford, and A. Eloyan. Randomness and statistical inference of shapes via the smooth Euler characteristic transform, 2022. arXiv:2204.12699.

[32] S. Micka. *Algorithms to Find Topological Descriptors for Shape Reconstruction and How to Search Them*. PhD thesis, Montana State University, 2020.

[33] K. V. Nadimpalli, A. Chattopadhyay, and B. Rieck. Euler characteristic transform based topological loss for reconstructing 3D images from single 2D slices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 571–579, 2023.

[34] M. J. Panik. *Fundamentals of convex analysis: duality, separation, representation, and resolution*, volume 24. Springer Science & Business Media, 2013.

[35] P. Pranav, H. Edelsbrunner, R. Van de Weygaert, G. Vegter, M. Kerber, B. J. Jones, and M. Wintraecken. The topology of the cosmic web in terms of persistent Betti numbers. *Monthly Notices of the Royal Astronomical Society*, 465(4):4281–4310, 2017.

[36] E. Richardson and M. Werman. Efficient classification using the euler characteristic. *Pattern Recognition Letters*, 49:99–106, 2014.

[37] A. H. Rizvi, P. G. Camara, E. K. Kandror, T. J. Roberts, I. Schieren, T. Maniatis, and R. Rabadan. Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. *Nature Biotechnology*, 35(6):551, 2017.

[38] A. Saadat-Yazdi, R. Andreeva, and R. Sarkar. Topological detection of alzheimer's disease using Betti curves. In *Interpretability of Machine Intelligence in Medical Image Computing, and TDA and Its Applications for Medical Data.*, pages 119–128. Springer, 2021.

[39] P. Schapira. Tomography of constructible functions. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 427–435. Springer, 1995.

[40] G. Singh, F. Mémoli, and G. E. Carlsson. Topological methods for the analysis of high dimensional data sets and 3D object recognition. *SPBG*, 91:100, 2007.

[41] K. Turner, S. Mukherjee, and D. M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 2014.

[42] S. Tymochko, E. Munch, J. Dunion, K. Corbosiero, and R. Torn. Using persistent homology to quantify a diurnal cycle in hurricanes. *Pattern Recognition Letters*, 2020.

[43] M. Usher and J. Zhang. Persistent homology and Floer–Novikov theory. *Geometry & Topology*, 20(6):3333–3430, 2016.

[44] R. Van De Weygaert, E. Platen, G. Vegter, B. Eldering, and N. Kruithof. Alpha shape topology of the cosmic web. In *2010 International Symposium on Voronoi Diagrams in Science and Engineering*, pages 224–234. IEEE, 2010.

[45] Y. Wang, H. Ombao, and M. K. Chung. Statistical persistent homology of brain signals. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1125–1129, 2019.

[46] G. Wilding, K. Nevenzeel, R. van de Weygaert, G. Vegter, P. Pranav, B. J. Jones, K. Efstathiou, and J. Feldbrugge. Persistent homology of the cosmic web–I. hierarchical topology in λCDM cosmologies. *Monthly Notices of the Royal Astronomical Society*, 507(2):2968–2990, 2021.

[47] L. Zhou. *Beyond Persistent Homology: More Discriminative Persistent Invariants*. PhD thesis, The Ohio State University, 2023.

[48] Z. Zhou, Y. Huang, L. Wang, and T. Tan. Exploring generalized shape analysis by topological representations. *Pattern Recognition Letters*, 87:177–185, 2017.

A Example Filtration with Six Descriptor Types

We consider the simplicial complex on four vertices given in Figure 7(a). In the e_1 direction we see four distinct heights of vertices, a, b, c, and d.

First, we describe what happens in $\rho(K)$ and $\tilde{\rho}(K)$. At height a, and then again at height b, we see connected components born. At height c, the homology of the sublevel set does not change, so no change is recorded in $\rho(K)$. However, a corresponding index filtration sees the connected component corresponding to first adding the vertex at c, which then immediately dies as we include the edge at height c. Thus, in $\tilde{\rho}(K)$, we have the point (c, c). For similar reasons, we see the point (d, d) in $\tilde{\rho}(K)$. Also at height d, our two connected components merge into a single connected component. It is a standard convention to choose the eldest component to survive, so we have the point (b, d) in both diagrams. We also see a cycle appear at height d, giving us the point (d, ∞) in both diagrams. Finally, since the connected component born at height a did not die, we have the point (a, ∞) in both diagrams.

Next, we describe what happens in $\beta(K)$ and $\tilde{\beta}(K)$. At the height a, only the Betti number in dimension zero changes, going from zero to one. Since the inclusion of this vertex increased Betti zero, we count the vertex as positive in $\tilde{\beta}_0(K)$. At the height b, again, only Betti zero changes, going from one to two, and we also count the corresponding vertex as positive for $\tilde{\beta}_0(K)$. At the height of c, no Betti number changes, and thus there is no event in $\beta(K)$. However, in a correpsonding index filtration, the inclusion of the vertex at height c increases β_0 by one, and the inclusion of the edge then reduces β_0 by one. This is recorded in $\tilde{\beta}(K)$ at c as an additional positive simplex (going from two total to three total), and our first negative simplex. We see similar behavior in dimension zero at the height of d. At height d we also see β_1 go from zero to one, which is recorded in the concise Betti function. In a corresponding index filtration, the inclusion of the second edge at height d increases β_1, and is thus recorded as positive.

Finally, we describe what happens in $\chi(K)$ and $\tilde{\chi}(K)$. At both heights a and b, the Euler characteristic of the sublevel sets increases by one. Since this is due to inclusions of vertices, which are even-dimensional simplices, both of these increases are recorded as even-dimensional in $\tilde{\chi}(K)$. At c, the Euler characteristic remains the same, so no change occurs in $\chi(K)$. In a corresponding index filtration, we see the vertex (an even-dimensional simplex) and an edge (odd-dimensional simplices), which are recorded in $\tilde{\chi}(K)$. Finally, the Euler characteristic at d changes

from two to zero, which is recorded directly in $\chi(K)$. An index filtration witnesses the appearance of one even and two odd simplices at height d, and this is recorded in $\tilde{\chi}(K)$.

B Faithfulness and Set Intersections

In this appendix, we explore reframing the definition of faithful in terms of set intersection. In Definition 2, we identify $\mathrm{D}(K,P)$ as faithful topological transform if, for any simplicial complex L, we have $\mathrm{D}(L,P) = \mathrm{D}(K,P)$ if and only if $K = L$. To unpack the equality $\mathrm{D}(L,P) = \mathrm{D}(K,P)$, recall

$$\mathrm{D}(K,P) := \{(p, \mathrm{D}(K,p))\}_{p\in P}.$$

Thus, $\mathrm{D}(L,P) = \mathrm{D}(K,P)$ if and only if for all $p \in P$, we have $\mathrm{D}(K,p) = \mathrm{D}(L,p)$. Then, $\mathrm{D}(K,P)$ is faithful if and only if

$$\bigcap_{p\in P} \left\{K' \subset \mathbb{R}^d \mid \mathrm{D}(K',p) = \mathrm{D}(K,p)\right\} = \{K\}. \quad (1)$$

From this perspective, we prove the following lemma providing a sufficient condition for finite faithful sets.

Lemma 13 (Sufficient Conditions for Finite Faithful Set) *Let K be a simplicial complex immersed in $\mathbb{R}^d$ and let* D *be a type of topological descriptor that can faithfully represent K. Suppose there exists a finite set of descriptors of type* D *that is faithful for K_0. Then, there exists a finite faithful set of descriptors of type* D *for K.*

Proof. Let P be a parameter set such that $\mathrm{D}(K,P)$ that is faithful for K, and let P_0 be a finite parameter set such that $\mathrm{D}(K,P_0)$ that is faithful for K_0.

Now, let B be the set of simplicial complexes that are indistinguishable from K using only parameter set P_0; that is,

$$B := \bigcap_{p\in P_0} \left\{K' \subset \mathbb{R}^d \mid \mathrm{D}(K',p) = \mathrm{D}(K,p)\right\} \quad (2)$$

Since P_0 is faithful for K_0, we know that $B \subseteq \{K' \mid K'_0 = K_0\}$, i.e., B is a subset of all simplicial complexes built out of the vertices of K. In particular, we note that this set is finite; since $|K_0|$ is finite, there are a finite number of simplicial complexes we can build over this set of vertices.

If $B = \{K\}$, we are done. Otherwise, since $D(K,P)$ is faithful for K, for each $L \neq K$ in B, there exists a direction $p_L \in P$ such that $\mathrm{D}(L,p_L) \neq \mathrm{D}(K,p_L)$. Let $P^* = P \cup \{p_L \mid L \in B\}$. Then, $\mathrm{D}(K,P^*)$ faithfully represents K. Furthermore, since P and B are finite, we also know that P^* is finite. □

C Zoo of Other Descriptor Types

In Section 2, we adopt a general definition of topological descriptor (Definition 1). In this appendix, we explore non-standard topological descriptors, and corresponding scenarios that may arise as a result of this generality. The descriptors presented here are not intended to be taken as anything that would necessarily make sense to use in practice, but rather, as a sort of zoo of examples to get a quick glance at the mathematical extremes and properties of the space of strength classes of topological descriptors.

First, we give an example of two distinct descriptor types that are in the same equivalence class of strength.

Example 1 *Consider the topological descriptor denoted $-\rho$ that takes a lower-star filtration in direction s, and produces $\rho(-s)$, the persistence diagram in direction $-s$. Although generally $\rho(s) \neq \rho(-s)$ (as multisets), a faithful set $\rho(K,S)$ has the same cardinality as the faithful set $-\rho(K,-S)$, and if a simplicial complex has no faithful set of type ρ, then it has no faithful set of type $-\rho$. Thus, $[\rho] = [-\rho]$.*

Next, we observe that many examples of topological descriptors are not capable of faithfully representing most simplicial complexes, such as the following.

Descriptor Type 1 (First Vertex) *Consider a descriptor D_V that returns (1) the coordinates of the first vertex (or vertices) encountered and (2) the cardinality of the vertex set, but no other information.*

If the filtrations are directional filtrations, then this descriptor is only faithful for convex point clouds. Any set of vertices that defines the corners of a convex region can be faithfully represented by this Descriptor Type 1. However, since no vertex interior to the convex hull nor any higher dimensional simplices are witnessed by any direction, this descriptor type is incapable of faithfully representing any other type of simplicial complex.

We can also construct descriptor types that are simply never able to form faithful sets.

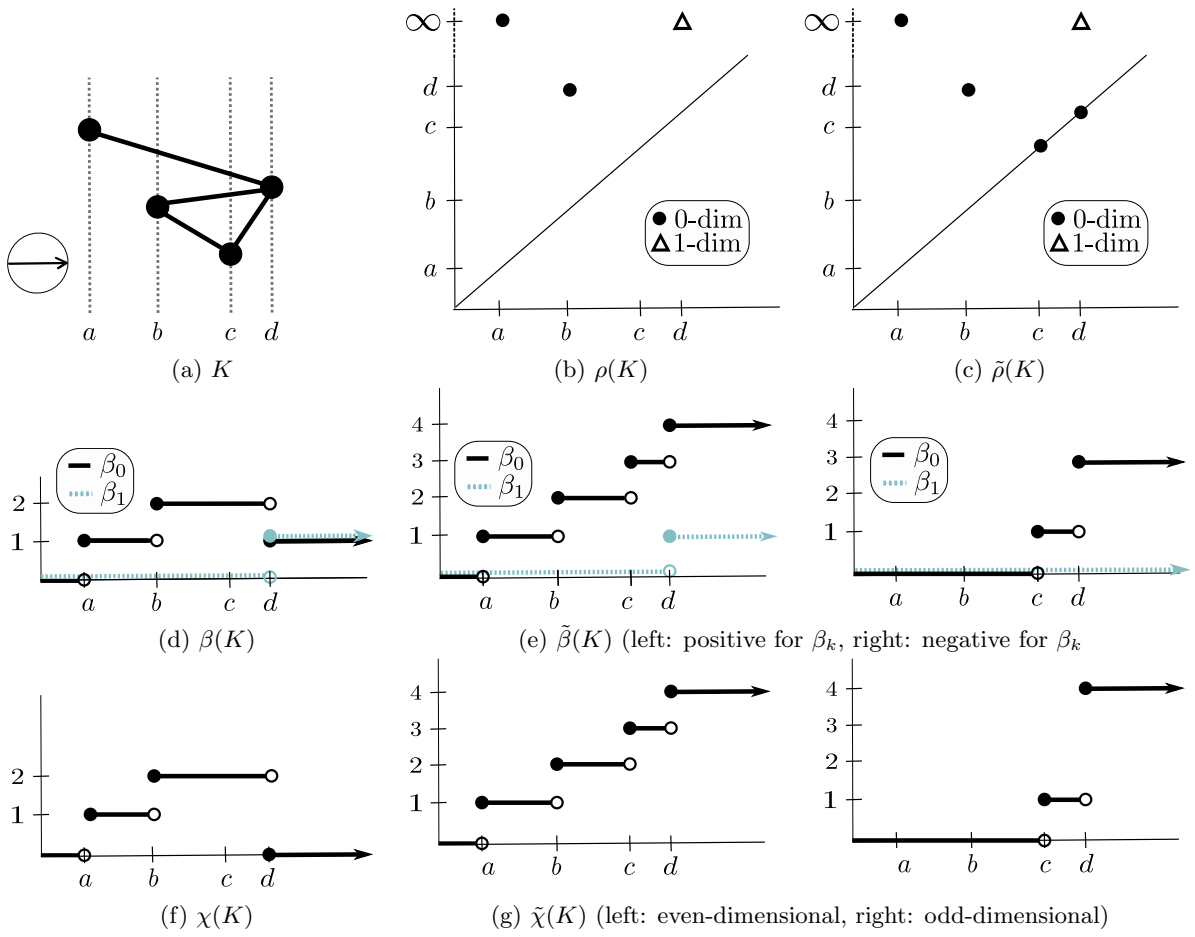
Descriptor Type 2 (Trivial) *Consider the trivial descriptor type D_0 that returns zero for all sublevel sets in a lower-star filtration.*

Although this trivial descriptor type is an invariant of any filtration, it can not faithfully represent any simplicial complex. Thus, $\Gamma(K, D_0) = \aleph_\top$ for all K. And so, in the space of all topological descriptors, Descriptor Type 3 is in the minimum strength class. We can (also trivially) construct a descriptor type that is in the maximum strength class.

Descriptor Type 3 (Filtration-Returning) *Consider the descriptor type D_{Filt} that returns the input filtration.*

Thus, a single descriptor of this type is always faithful for a simplicial complex.

Finally, we can find instances of topological descriptors that are able to faithfully represent a simplicial complex, but with a set no smaller than uncountably infinite.



(a) K (b) $\rho(K)$ (c) $\tilde{\rho}(K)$

(d) $\beta(K)$ (e) $\tilde{\beta}(K)$ (left: positive for β_k, right: negative for β_k

(f) $\chi(K)$ (g) $\tilde{\chi}(K)$ (left: even-dimensional, right: odd-dimensional)

Figure 7: Six descriptors corresponding to the lower-star filtration in the direction indicated by the arrow of the simplicial complex in (a).

Descriptor Type 4 (Indicator) *Let K be a simplicial complex immersed in $\mathbb{R}^d$ let $D_{\mathbb{R}}$ be a descriptor type parameterized by $\mathbb{R}^d$ that is constant over a filtration and is defined by*

$$D(K,x) = \begin{cases} 1 & \text{if } x \in |K| \\ 0 & \text{else.} \end{cases}$$

Note that then, $D(K,\mathbb{R}^d)$ is the (only) minimum faithful set for K, and so $\Gamma(K, D_{\mathbb{R}}) = \aleph_1$ for all K. Thus, the (minimal) strength class of Descriptor Type 4 is greater than the strength class of the trivial descriptor in Descriptor Type 3, and there are no strength equivalence classes between them.

We now know the space of strength classes of topological descriptors has a minimum and maximum, and we have identified a second smallest descriptor type; is it a total order? The following example shows that it is not; there do indeed exist incomparable descriptor types.

Lemma 14 (Incomparable Strength Classes) *There exist incomparable strength classes of topological descriptor types.*

Proof. Let D_V denote Descriptor Type 1. That is, given a direction s, D returns: (1) the coordinates of the lowest vertex (or vertices) in direction s, and (2) the cardinality of the vertex set. We compare D_V with verbose persistence diagrams. Let $v_1 = (0,0)$ and $v_2 = (0,1)$.

First, consider the simplicial complex $K = \{v_1\}$. Then, regardless of direction, a single descriptor of is faithful for K. However, since K is in $\mathbb{R}^2$, any faithful set of augmented persistence diagrams must have at least two linearly independent directions to recover both coordinates of K.

Next, consider the simplicial complex $K' = \{v_1, v_2\}$. No set of descriptors of type D_V is faithful for K' (it cannot distinguish between K' and the simplicial complex consisting of the two disconnected vertices v_1 and v_2 without an edge). However, two augmented persistence diagrams suffice to form a faithful set for K'; for example, using the standard basis vectors $\{e_1, e_2\}$ as the set. Thus, if $\mathrm{D}_V(K, S_{\mathrm{D}_V})$ and $\tilde{\rho}(K, S_{\tilde{\rho}})$ are both minimal faithful sets, we see that $|S_{\mathrm{D}_V}| < |S_{\tilde{\rho}}|$ but $|S_{\mathrm{D}_V}| > |S_{\tilde{\rho}}|$. Thus, although we have shown $[\mathrm{D}_V] \neq [\tilde{\rho}]$, they are incomparable. □

Finally, we give a lemma that shows an impact of not adopting the general position assumption for the vertices. Specifically, the general position assumption we use throughout this paper is:

General Position 1 *A simplicial complex K immersed in $\mathbb{R}^d$ is in* general position *if, for all subsets $V \subseteq K_0$ with $|V| \leq d+1$ is affinely independent*

Removing this assumption, we obtain:

Lemma 15 (Concise Equality) *Without Assumption 1, the strength equivalence classes of the three concise topological descriptor types from Section 3 are all equal.*

Proof. Let $D \in \{\chi, \beta, \rho\}$. We must consider faithful sets of such descriptors for an arbitrary simplicial complex K immersed in $\mathbb{R}^d$ (that may not be in general posistion). The argument differs depending on if K is a vertex set, or contains at least one edge; we consider each case.

First, suppose $n_1 = 0$. Then, K has no edges and is a vertex set, meaning each vertex is a maximal simplex of K. Then, by Corollary 1, faithful sets of type D must include descriptors from at least $d+1$ directions. By Lemma 7, the envelopes of each vertex must be zero-dimensional. Since the only zero-dimensional convex sets are singleton points, the envelope of each vertex contains that vertex and nothing else.

Let S be a set of $d+1$ directions such that the envelope of each vertex is zero-dimensional (note that such a set exists, for example, the standard basis directions (e_i), and the negative diagonal direction, $-1/\sqrt{d}(1, 1, \ldots, 1)$). Consider a single direction, $s \in S$ and $a \leq b \in \mathbb{R}^d$. If no event occurs in $\mathrm{D}(s)$ between heights a and b, we know no connected component of K has its lowest vertex (or vertices) with respect to s in the range from a to b. Then, from $\mathrm{D}(s)$, we identify that K has n_0 connected components, and we know their starting heights with respect to the s direction. In other words, we know these connected components are contained in the (closed) upper half-spaces defined by s.

Each additional direction in S gives us more information about these n_0 connected components, and, just like s, each additional direction provides an additional upper half-space in which we know the connected component is contained; each connected component must lie in the intersection of these half-spaces. Since we use a total of $d+1$ directions, chosen so that the envelope of each vertex, $\mathcal{E}_v^S$ (the intersection of half-spaces), is zero-dimensional, we conclude each connected component is zero-dimensional.

That is, we know the exact location of each vertex by identifying its envelope. Thus, minimal faithful sets of type D have cardinality of exactly $d+1$, meaning the infimums considered in Definition 7 are $\Gamma(K, D) = d+1$ for such K.

Next, suppose $n_1 > 1$. We show no set of descriptors of type D can faithfully represent K. Let τ be an edge in K and construct another complex L by starting with K and taking the barycentric subdivision of τ and all simplices containing τ. Then, since $|K| = |L|$ the Euler characteristics/Betti numbers/homology throughout the filtrations of K or L agree. That is, for every direction s, $\mathrm{D}(K, s) = \mathrm{D}(L, s)$. Since this is true for every s, the descriptor type D is incapable of forming a faithful set for K, meaning the infimums considered in Definition 7 are $\Gamma(K, D) = \aleph_\top$ for such K.

We have shown that, for each K without a general position assumption, we have $\Gamma(K, \chi) = \Gamma(K, \beta) = \Gamma(K, \rho)$. Thus, when removing the general position assumption, we find $[\chi] = [\beta] = [\rho]$. □

Fortunately, as shown in Section 5, the relations among concise descriptors becomes more interesting when we assume general position. This also has the benefit of reflecting the general position assumptions that are often taken in practical applications.

D Omitted Proofs and Lemmas

In this appendix, we provide proofs that were omitted from the main text.

D.1 Proof from Section 4

The following lemma is stated, but not proven, in Section 4.

Lemma 1 *Let A and B be two topological descriptor types. If B is reducible to A, we have $[A] \preceq [B]$.*

Proof. Let K be a simplicial complex. Define the sets $W_A := \{P \text{ s.t. } A(K, P) \text{ is faithful}\}$ and $W_B := \{P \text{ s.t. } B(K, P) \text{ is faithful}\}$. For each $P \in W_A$, by definition, $A(K, P)$ is faithful. Since B is reducible to A, this also means $B(K, P)$ is faithful, and so P is also in W_B. Hence, $W_A \subseteq W_B$. Hence, $\inf_{P \in W_A} |P| \geq \inf_{P \in W_B} |P|$. Note that these are exactly the infimums in Definition 7, and so, we have $[A] \preceq [B]$. □

The following lemmas are referenced, but not explicitly stated, in Section 4.

Lemma 16 *The relation $=$ is an equivalence relation, and the relation $\preceq$ is well-defined on sets of strength equivalence classes.*

Proof. When we compare infimums in Definition 7, we compare values in $\mathbb{N} \cup \{\aleph_0, \aleph_1, \aleph_\top\}$. The relation $\leq$ on values in this set is reflexive, antisymmetric, and transitive. The relation $=$ on this set is reflexive, symmetric, and transitive. The result follows. □

D.2 Proofs from Section 5

We prove two lemmas that were originally stated in Section 5.

Lemma 2 $[\chi] \preceq [\beta] \preceq [\rho]$ *and* $[\tilde{\chi}] \preceq [\tilde{\beta}] \preceq [\tilde{\rho}]$.

Proof. The proof follows directly from a reduction argument. We can reduce any $\rho(s)$ to $\beta(s)$ by "forgetting" the relationship between birth and death events. We can then reduce $\beta(s)$ to $\chi(s)$ by taking the alternating sum of points from $\beta(s)$. A nearly identical argument shows the relationship between verbose versions of these descriptors. □

The reductions described above are well-known, and are observed in other work; for example [7, Prop 4.13] points out the reduction from an Euler characteristic function to a persistence diagram.

Lemma 3 *$[\chi] \preceq [\tilde{\chi}]$, $[\beta] \preceq [\tilde{\beta}]$, and $[\rho] \preceq [\tilde{\rho}]$.*

Proof. Each verbose descriptor has a clear reduction to its concise counterpart. A verbose persistence diagram becomes a concise persistence diagram by removing all on-diagonal points. Verbose Betti functions and verbose Euler characteristic functions become concise if we subtract their second coordinates from their first coordinates. Then, by Lemma 1, we have the desired relations. □

Lemma 4 *For $\mathrm{D} \in \{\chi, \beta, \rho\}$ and $\tilde{\mathrm{D}} \in \{\tilde{\chi}, \tilde{\beta}, \tilde{\rho}\}$, we have $[\mathrm{D}] \neq [\tilde{\mathrm{D}}]$.*

Proof. To show inequality of strength classes, we find a simplicial complex for which minimum faithful sets of type D and $\tilde{\mathrm{D}}$ have different cardinalities. Let K be the simplicial complex that is a single edge in $\mathbb{R}^2$ with vertex coordinates $(1, 1)$ and $(1, 2)$. See Figure 8 for this complex, and an illustration for the specific case $\tilde{\mathrm{D}} = \tilde{\rho}$. In the direction $e_1 = (1, 0)$, if $\tilde{\mathrm{D}} = \tilde{\rho}$, we see an instantaneous birth/death and an infinite birth in degree zero. If $\tilde{\mathrm{D}} = \tilde{\beta}$, we see two positive simplices and one negative simplex for Betti zero. If $\tilde{\mathrm{D}} = \tilde{\chi}$, we see two even simplices and one odd simplex. This all occurs at height 1, and there are no other events, which, is only explainable by the presence of a single edge. From $\tilde{\mathrm{D}}(e_2)$, we see a non-instantaneous and instantaneous event at heights 1 and 2, respectively, which give us the y-coordinates of our two vertices. Then, K is the only complex that could have generated both $\tilde{\mathrm{D}}(e_1)$ and $\tilde{\mathrm{D}}(e_2)$, i.e., the set $\tilde{\mathrm{D}}(K, \{e_1, e_2\})$ is faithful.

Next, consider the descriptor type D. For any $s \in \mathbb{S}^1$, $\mathrm{D}(s)$ contains exactly one event; if the lowest vertex of K with respect to s has height a in direction s, then $\mathrm{D}(s)$ records a change in homology/Betti number/Euler characteristic at height a and records no other changes. Thus, $\mathrm{D}(s)$ can only give us information about one coordinate of the vertex set of K at a time, corresponding to whichever vertex is lowest in direction s. However, K has three relevant coordinates; namely, $x = 1$, $y = 1$, and $y = 2$ meaning it is not possible for any faithful set of type D to have size less than three. Thus, since $3 \neq 2$, we have shown $[\mathrm{D}] \neq [\tilde{\mathrm{D}}]$, as desired. □

The specific inequality $[\chi] \neq [\tilde{\rho}]$ is also implied by [32, Thm. 10].

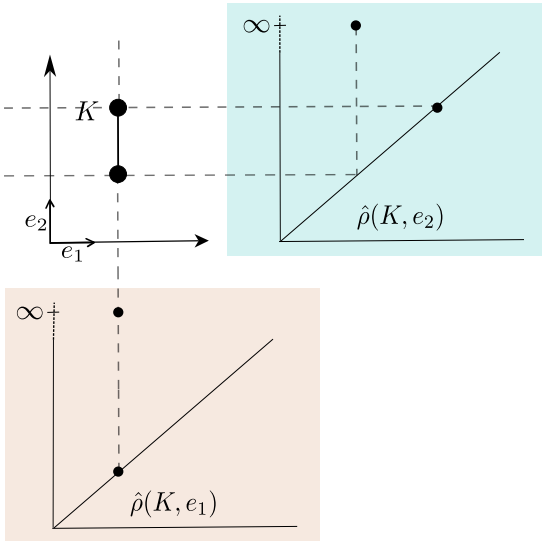


Figure 8: The simplicial complex considered in the proof of Lemma 4 as well as the verbose diagrams in directions e_1 and e_2. Note that $\tilde{\rho}(K, \{e_1, e_2\})$ is indeed a faithful set, since we can recover the coordinates of both vertices ($\tilde{\rho}(K, e_1)$ tells us the x-coordinates and $\tilde{\rho}(K, e_2)$ tells us the y-coordinates), as well as determine there is only a single edge present (there is only one instantaneous zero-dimensional point in each verbose diagram). The concise versions of these diagrams do not have on-diagonal points, and each only contain a single point at ∞. This is true for concise diagrams corresponding to any direction.

D.3 Proofs from Section 6

We provide proofs of Lemmas 7 and 10.

Lemma 7 (Envelopes for Faithful Concise Sets) *Let K be a simplicial complex immersed in $\mathbb{R}^d$, let $D \in \{\chi, \beta, \rho\}$, and let $S \subseteq \mathbb{S}^{d-1}$ so that $\mathrm{D}(K, S)$ is faithful. Then, for any maximal simplex σ in K, the dimension of $\mathcal{E}_\sigma$ equals the dimension of σ.*

Proof. Let k be the dimension of σ, and let c be the dimension of $\mathcal{E}_\sigma$. First, we observe that since σ is contained in $\mathcal{E}_\sigma$, we must have $k \leq c$. The claim is trivial when $k = d$, so we proceed with the case $k < d$ and assume, by way of contradiction, that $k < c$.

We claim that in this case, 1) at every interior point of σ, there is a vector normal to σ that ends in the interior of $\mathcal{E}_\sigma$, and 2) letting p denote the endpoint of such a vector, p is higher than the lowest vertex of σ with respect to each $s \in S$.

The first part of the claim, 1), is true since otherwise, $\mathcal{E}_\sigma$ would be k-dimensional. 2) is true since each halfspace defining $\mathcal{E}_\sigma$ contains the lowest vertex (or vertices) of σ with respect to the corresponding direction. Thus, since p is in the interior of $\mathcal{E}_\sigma$, it must be higher than this lowest vertex (or vertices) with respect to the corresponding direction.

Now consider the simplicial complex K', defined as having all the simplices of K in addition to the simplex formed by taking the geometric join of p and σ, i.e., the simplex $p * \sigma$. We claim that, for any $s \in S$, we have $\mathrm{D}(K', s) = \mathrm{D}(K, s)$. First, we note that since $p * \sigma$ deformation retracts onto σ, K' has the same homology as K. Next, we observe that $\mathrm{D}(K', s)$ and $\mathrm{D}(K, s)$ cannot differ by more than a connected component birth/death; higher dimensional differences would require more than the join of a point with an existing face.

Finally, since p is higher than the lowest vertex of σ with respect to any direction $s \in S$, the simplex $p * \sigma \in K'$ does not correspond to any connected component birth or death in $\mathrm{D}(K', s)$ that was not present in $\mathrm{D}(K, s)$. Thus, we have shown

$\mathrm{D}(K', S) = \mathrm{D}(K, S)$. This contradicts the assumption that $\mathrm{D}(K, S)$ is faithful, so we must have $k = c$. □

The next statement we wish to prove is Lemma 10. We first establish the following lemma.

Lemma 17 *Consider a pair of nested triangles as in Figure 9. Then, angle A is larger than θ, $\phi - B$, and $\psi - C$.*

Proof. Adding angles in the larger triangle, we see $\theta + \phi + \psi = \pi$. Then,

$$\begin{aligned}
\theta + (\phi - B) + B + (\psi - C) + C &= \pi \\
A + \theta + (\phi - B) + B + (\psi - C) + C &= A + \pi \\
(A + B + C) + \theta + (\phi - B) + (\psi - C) &= A + \pi \\
\pi + \theta + (\phi - B) + (\psi - C) &= A + \pi \\
\theta + (\phi - B) + (\psi - C) &= A.
\end{aligned}$$

All the terms in the last line are positive, meaning A is larger than θ, $\phi - B$, and $\psi - C$. □

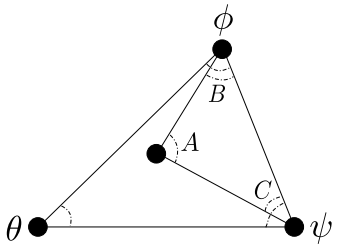


Figure 9: Nested triangles as discussed in Lemma 17

Now, we are armed with the tools needed to prove the following lemma.

Lemma 10 (Clothespin Representability) *Let K be as in Construction 1, and suppose that $\tilde{\rho}(K, S)$ is faithful. Then, there is some $s \in S$ so that the angle formed between s and the x-axis lies in the region*

$$W = [\alpha_{3,2} - \pi/2, \alpha_{3,4} - \pi/2] \cup [\alpha_{3,2} + \pi/2, \alpha_{3,4} + \pi/2].$$

Proof. Let K' be a simplicial complex immersed in $\mathbb{R}^2$ with the same vertex set as K, but with edges $[v_1, v_4]$ and $[v_2, v_3]$ (see Figure 5b). Recall that, since $\tilde{\rho}(K, S)$ is faithful, by definition, the set S must contain some direction s so that $\tilde{\rho}(K, s) \neq \tilde{\rho}(K', s)$.

Each vertex corresponds to either a birth event or an instantaneous event depending on the direction of filtration. We proceed by considering each vertex v_i individually and determining subsets $R_i \subset \mathbb{S}^1$ such that, whenever $s \in R_i$, the event at $s \cdot v_i$ is different when filtering over K versus K', but for $s_* \notin R_i$, the type of event at $s_* \cdot v_i$ is the same between the two graphs. Figure 10 shows these regions, and in what follows, we define them precisely.

First, consider v_1. By Observation 1, $v_1 \in K$ corresponds to a birth event for all directions in the interval $B = (\alpha_{1,2} - \pi/2, \alpha_{1,2} + \pi/2)$ and $v_1 \in K'$ corresponds to a birth event for all directions in the interval $B' = (\alpha_{1,4} - \pi/2, \alpha_{1,4} + \pi/2)$. Then, we write $R_1 = (B \setminus B') \cup (B' \setminus B)$, which is the wedge-shaped region such that for any $s \in R_1$, the type of event associated to $v_1 \in K$ and $v_1 \in K'$ differ, meaning $\tilde{\rho}(K, s) \neq \tilde{\rho}(K', s)$.

Using this same notation, identify the wedge shaped region R_i for vertex $i \in [2, 3, 4]$ such that any direction from R_i generates verbose persistence diagrams that have different event types at the height of vertex v_i when filtering over K versus K'. Similar arguments for $i \in [2, 3, 4]$ give us the complete list;

$$\begin{aligned}
R_1 &= (\alpha_{1,2} - \pi/2, \alpha_{1,4} - \pi/2] \cup [\alpha_{1,2} + \pi/2, \alpha_{1,4} + \pi/2) \\
R_2 &= (\alpha_{2,3} - \pi/2, \alpha_{2,1} - \pi/2] \cup [\alpha_{2,3} + \pi/2, \alpha_{2,1} + \pi/2) \\
R_3 &= (\alpha_{3,2} - \pi/2, \alpha_{3,4} - \pi/2] \cup [\alpha_{3,2} + \pi/2, \alpha_{3,4} + \pi/2) \\
R_4 &= (\alpha_{1,4} - \pi/2, \alpha_{3,4} - \pi/2] \cup [\alpha_{1,4} + \pi/2, \alpha_{3,4} + \pi/2)
\end{aligned}$$

Let $W = \cup_{i=1}^{4} R_i$. Then, for any $s \in W$, we have $\tilde{\rho}(K, s) \neq \tilde{\rho}(K', s)$, and for any $s_* \in W^C$, we have $\tilde{\rho}(K, s_*) = \tilde{\rho}(K', s_*)$.

Finally, we claim that W is the closure of R_3, denoted $\overline{R}_3$, i.e., exactly the region described in the lemma statement. This is a direct corollary to Lemma 17; the angles swept out by each regions correspond to the angles formed by pairs of edges in K and K'; in particular, the angle $\measuredangle v_2 v_3 v_4$ is the largest and geometrically contains the others. This means the extremal boundaries over all R_i's are formed by the angles $\alpha_{2,3} \pm \pi/2$ and $\alpha_{3,4} \pm \pi/2$, the defining angles of R_3. Each of these four angles appears as an included endpoint for some R_i, so $R_1, R_2, R_4 \subseteq \overline{R}_3 = W$ (Figure 10) and we have shown our claim. □

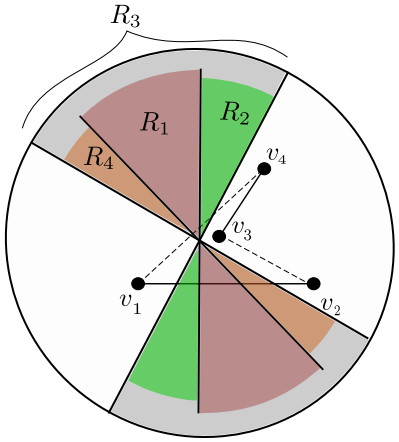


Figure 10: The regions described in the proof of Lemma 10, with additional shading in the interior of the sphere of directions to aid in visibility. K is shown as solid black edges and K' as dashed edges. For any lower-star filtration in a direction contained in R_i, the event at vertex v_i differs when considering K or K', thus, such directions are able to distinguish K from K'. Note that any direction outside the regions of observability (i.e., the non-shaded portions of the circle) is not able to distinguish K from K'.

On the crossing number of symmetric configurations

Bernardo Ábrego* Silvia Fernández-Merchant†

Abstract

The *crossing number* of a finite set P of points in general position in the plane is the number of pairs of segments joining points in P that cross each other. The minimum crossing number over all sets P of n points in general position is known as the *rectilinear* or *geometric crossing number* of the complete graph K_n. We bound this crossing number when the minimum is restricted to m-fold symmetric configurations of points in the plane. Our bounds are tight for even symmetry.

1 Introduction

In a *drawing* of a simple graph G (on the plane) the vertices of G are represented by points in the plane and the edges are curves joining the vertices. The number of crossings in a drawing D of a graph is denoted by $\mathrm{Cr}(D)$. The minimum number of crossings over all drawings of a graph G is denoted by $\mathrm{cr}(G) := \min\{\mathrm{Cr}(D) : D \text{ drawing of } G\}$. Note that we are counting pairs of edges that cross, so if k edges cross at the same point, this point represents $\binom{k}{2}$ crossings.

In the 1940s, Paul Turán [16] considered this problem for complete bipartite graphs and due to its history [13] the problem is now known as the *Brick Factory Problem*. Independently, Harary and Hill studied the problem for complete graphs [14] and conjectured that

$$\mathrm{cr}(K_n) = H(n) := \frac{1}{4}\left\lfloor \frac{n}{2}\right\rfloor \left\lfloor \frac{n-1}{2}\right\rfloor \left\lfloor \frac{n-2}{2}\right\rfloor \left\lfloor \frac{n-3}{2}\right\rfloor .$$

This conjecture remains open, however in the last 10 years a lot of progress has been made [2, 3, 9, 11]. These results were based on tools that were first developed for a related problem, where the minimum number of crossing is taken over a smaller family of drawings. Namely, those where the edges are straight line segments. This is known as a *geometric drawing*. In this context, we define the *geometric crossing number* of a graph G, denoted by $\overline{\mathrm{cr}}(G)$ (also known as *rectilinear crossing number*), as the minimum number of crossings over all geometric drawings of G. The geometric crossing number of the complete graph, $\overline{\mathrm{cr}}(K_n)$ has also been extensively studied [1, 7, 10, 11, 12]. Note that $\mathrm{cr}(K_n) \le \overline{\mathrm{cr}}(K_n)$. In fact, it is known that $\mathrm{cr}(K_n)$ and $\overline{\mathrm{cr}}(K_n)$ are different even for $n = 8$. The latest improvements on the geometric variation of the problem use the following approach.

Let P be a finite set of points in general position in the plane. We say that a pair of points $\{p, q\} \subset P$ is a *k-edge* of P if the line spanned by p and q separates k points of P from the rest. An *at-most-k-edge* is a j-edge for some $j \le k$. We denote the number of k-edges and at-most-k-edges of P by $E_k(P)$ and $E_{\le k}(P)$, respectively. In 2004, a breakthrough was made by relating the crossing number of a geometric drawing to the number of k-edges determined by its set of vertices [6, 15].

Theorem 1 (Crossing number identity [6, 15]) *Let D be a geometric drawing of the complete graph K_n with set of vertices P. Then*

$$\begin{aligned}\mathrm{Cr}(D) &= 3\binom{n}{4} - \sum_{k=0}^{\lfloor n/2\rfloor -1} E_k(P)k(n-2-k)\\ &= \sum_{k=0}^{\lfloor n/2\rfloor -2} (n-2k-3)E_{\le k}(P) - \frac{3}{4}\binom{n}{3}\\ &\qquad + (1+(-1)^{n+1})\frac{1}{8}\binom{n}{2}.\end{aligned}$$

In contrast to the original problem, there is no conjectured value for $\overline{\mathrm{cr}}(K_n)$. Recently, the case when the minimum is restricted to the family of centrally symmetric drawings, denoted by $\overline{\mathrm{cr}}_{\mathrm{cs}}(K_{2n})$, was considered in [5]. There, they proved what is probably the first identity for the rectilinear crossing number of a large family of geometric drawings of K_n.

Theorem 2 (Centrally symmetric identity [5]) *For any positive integer n, we have*

$$\overline{\mathrm{cr}}_{\mathrm{cs}}(K_{2n}) = 2\binom{n}{4} + \binom{n}{2}^2.$$

Their proof was based on Theorem 1 together with the following bound on the number of at most k-edges.

Lemma 3 *For any centrally symmetric set P of $2n$ points in general position in the plane and for any $k < n-1$,*

$$E_{\le k}(P) \ge 4\binom{k+2}{2}.$$

*California State University, Northridge, 18311 Nordhoff St, Northridge, CA, 91330, USA. bernardo.abrego@csun.edu, supported by a 2024 RSP CSUN Campus Funding Initiative

†California State University, Northridge, 18311 Nordhoff St, Northridge, CA, 91330, USA. silvia.fernandez@csun.edu, supported by a 2024 RSP CSUN Campus Funding Initiative

In this paper, we consider configurations of points with *m-fold (rotational) symmetry*, that is, sets P for which there is a $\frac{2\pi}{m}$-rotation $\mathcal{R}$ such that $\mathcal{R}(P) = P$ (the centrally symmetric case corresponds to $m = 2$). In this case, the *m-rotational orbit* of a point $x \in P$ is the set of points $[x] := \{x, \mathcal{R}(x), \mathcal{R}^2(x), \ldots, \mathcal{R}^{m-1}(x)\} \subset P$.

In Theorem 6, we provide improved lower bounds on the number of at most-k-edges for this symmetric case. In turn, using Theorem 1, these bounds provide improved lower bounds on the crossing number restricted to m-fold symmetric configurations, denoted by sym-$\overline{\text{cr}}_m(K_n)$. We finally show that these bounds are tight when m is even, settling the crossing number of symmetric configurations with even symmetry as follows:

Theorem 4 *For positive integers m and n such that m is even and $m|n$, we have*

$$\text{sym-}\overline{\text{cr}}_m(K_n) = \frac{1}{2}\binom{n}{4} + \frac{1}{2}\binom{n/2}{2} + \frac{n^2}{6}\binom{m/2-1}{2}.$$

Note that Theorem 4 agrees with Theorem 2 when $m = 2$. In contrast to the best known bounds for the general geometric case (lower bound [8], upper bound [7, 11])

$$0.379972 \leq \lim_{n\leftarrow\infty} \frac{\overline{\text{cr}}(K_n)}{\binom{n}{4}} \leq 0.380449,$$

in the m-fold symmetric case with m even we have,

$$\lim_{n\leftarrow\infty} \frac{\text{sym-}\overline{\text{cr}}_m(K_n)}{\binom{n}{4}} = \lim_{n\leftarrow\infty} \frac{\frac{1}{2}\binom{n}{4} + \frac{1}{2}\binom{n/2}{2} + \frac{n^2}{6}\binom{m/2-1}{2}}{\binom{n}{4}} = \frac{1}{2}.$$

2 Bounds on k-edges

Lemma 5 *Let P be a set of n points in general position in the plane and $0 \leq k \leq \lceil n/2\rceil - 2$ an integer. Suppose P is m-fold symmetric and consider a point x in the convex hull of P. Let $P' = P - [x]$, where $[x] \subset P$ is the m-rotational orbit of x. Then*

$$E_{\leq k}(P) \geq E_{\leq k-\lceil m/2\rceil}(P') + 2m(k+1) - m\cdot\min(k+1, \lceil m/2\rceil - 1).$$

(Note that $E_{\leq j}(P) = 0$ for negative values of j.)

Proof. Let P' be the set obtained from P by removing the m points of $[x]$. Note that P' is an m-fold symmetric set of $n - m$ points. Consider an edge e of P (any line passing through two points of P) and let ℓ be the line parallel to e passing through p. Note that ℓ divides $[x]$ in almost half, leaving at most $\lceil m/2\rceil$ points on each side. Since the smaller side of e is contained on one of the two sides of ℓ, then e leaves at most $\lceil m/2\rceil$ points of $[x]$ on its smaller side. This means that if e' is a j-edge of P', then e' is a $(\leq j + \lceil m/2\rceil)$-edge of P. Therefore, for any $k \leq \lceil n/2\rceil - 2$ any $(\leq k - \lceil m/2\rceil)$-edge of P' is a $\leq k$-edge of P. And none of these $\leq k$-edges is incident on points of x.

Because the points of $[x]$ are on the convex hull of P, each of them participates in exactly two j-edges for each $0 \leq j \leq \lceil n/2\rceil - 2$. This gives at most $2m(k+1)$ at-most-k-edges of P incident on points of $[x]$. However, some of these edges may be incident to two points of $[x]$: as many as $m(k+1)$ when $k \leq \lceil m/2\rceil - 1$ and as many as $m(\lceil m/2\rceil - 1)$ (all pairs of $[x]$, except its halving lines when m is even) when $k \geq \lceil m/2\rceil - 1$. In summary, there are at least $E_{\leq k-\lceil m/2\rceil}(P')$ at-most-k-edges of P not incident on $[x]$, and at least $2m(k+1) - m\cdot\min(k+1, \lceil m/2\rceil - 1)$ at-most-k-edges of P incident on $[x]$. □

Theorem 6 *Let m be a positive integer. For any set P of n points with m-fold symmetry and for any $0 \leq k \leq \lceil n/2\rceil - 2$, we have that $E_{\leq k}(P) \geq$*

$$\begin{cases} 4\left(1 - \frac{1}{m+1}\right)\left(\binom{k+2}{2} - \binom{r+1}{2}\right) + mr & \text{if } m \text{ is odd,} \\ 4\left(\binom{k+2}{2} - \binom{r+1}{2}\right) + mr & \text{if } m \text{ is even.} \end{cases} \quad (1)$$

where $k + 1 = r \bmod \lceil m/2\rceil$.

Proof. If $0 \leq k \leq \lceil m/2\rceil - 2$, then $r = k + 1$ and $\min(k+1, \lceil m/2\rceil - 1) = k + 1$. Then, by Lemma 5, $E_{\leq k}(P) \geq 0 + m(k+1) = mr$ matching the right-hand-side of (1).

If $k = \lceil m/2\rceil - 1$, then $r = 0$. By Lemma 5,

$$\begin{aligned} E_{\leq k}(P) &\geq 0 + 2m(k+1) - m\cdot\min(k+1, k) = m(k+2) \\ &= \frac{2m}{k+1}\binom{k+2}{2} = \frac{2m}{\lceil m/2\rceil}\binom{k+2}{2} \\ &= \begin{cases} 4\left(\frac{m}{m+1}\right)\binom{k+2}{2} & \text{if } m \text{ is odd,} \\ 4\binom{k+2}{2} & \text{if } m \text{ is even,} \end{cases} \end{aligned}$$

matching the right-hand-side of (1).

We prove the result by induction on n. The inequality holds for $n = m$ because a k-edge of a set of m points is a $(\leq \lceil m/2\rceil - 1)$-edge (proved above). Now let $n \geq m$ and assume that the inequality holds for any m-fold symmetric set of n points and any $k < \lceil n/2\rceil - 2$. Consider any m-fold symmetric set P of $n+m$ points. Let x be a point on its convex hull and P' be the set obtained from P by removing the m points of $[x]$. Note that P' is an m-fold symmetric set of n points and so it satisfies the induction hypothesis. We already proved the result holds for $k \leq \lceil m/2\rceil - 1$. Assume that $k \geq \lceil m/2\rceil$. Note that $(k - \lceil m/2\rceil) + 1 \equiv k + 1 \equiv r \pmod{\lceil m/2\rceil}$. By

Lemma 5,

$$E_{\le k}(P) \ge E_{\le k-\lceil m/2\rceil}(P') + 2m(k+1) - m\cdot\min(k+1, \lceil m/2\rceil - 1) \ge m\left(2k+3-\left\lceil\frac{m}{2}\right\rceil\right) + \begin{cases} 4\left(\frac{m}{m+1}\right)\left(\binom{k+2-\lceil m/2\rceil}{2} - \binom{r+1}{2}\right) + mr & \text{if } m \text{ is odd,} \\ 4\left(\binom{k+2-\lceil m/2\rceil}{2} - \binom{r+1}{2}\right) - mr & \text{if } m \text{ is even.} \end{cases} \quad (2)$$

Since

$$\binom{k+2-\lceil m/2\rceil}{2} = \binom{k+2}{2} - \frac{1}{2}\left\lceil\frac{m}{2}\right\rceil\left(2k+3-\left\lceil\frac{m}{2}\right\rceil\right),$$

the right-hand-side of (2) equals

$$\begin{cases} 4\left(\frac{m}{m+1}\right)\left(\binom{k+2}{2} - \binom{r+1}{2}\right) + mr \\ \quad + \left(2k+3-\left\lceil\frac{m}{2}\right\rceil\right)\left(m - 2\cdot\frac{m}{m+1}\cdot\frac{m+1}{2}\right) & \text{if } m \text{ is odd,} \\ 4\left(\binom{k+2}{2} - \binom{r+1}{2}\right) + mr + \left(2k+3-\left\lceil\frac{m}{2}\right\rceil\right) & \text{if } m \text{ is even,} \end{cases} = \begin{cases} 4\left(1-\frac{1}{m+1}\right)\left(\binom{k+2}{2} - \binom{r+1}{2}\right) + mr & \text{if } m \text{ is odd,} \\ 4\left(\binom{k+2}{2} - \binom{r+1}{2}\right) + mr & \text{if } m \text{ is even.} \end{cases}$$

□

3 Exact crossing number for even symmetry

In this section, we prove Theorem 4. Throughout this section, m is an even integer. A construction achieving the stated crossing number is presented in Section 3.1. To prove that sym-$\overline{\text{cr}}_m(K_n) \ge \frac{1}{2}\binom{n}{4} + \frac{1}{2}\binom{n/2}{2} + \frac{1}{6}\binom{m/2-1}{2}n^2$, we use Theorem 1 followed by Theorem 6.

Let D be a geometric m-fold symmetric drawing of the complete graph K_n with set of vertices P. For $0 \le k \le n/2-2$, define r_k so that $k+1 = r_k \bmod (m/2)$. Then

$$cr(P) = \sum_{k=0}^{n/2-2}(n-2k-3)E_{\le k}(P) - \frac{3}{4}\binom{n}{3} + (1+(-1)^{n+1})\frac{1}{8}\binom{n}{2} \ge \sum_{k=0}^{n/2-2}(n-2k-3)\left(4\binom{k+2}{2} - 4\binom{r_k+1}{2} + mr_k\right) - \frac{3}{4}\binom{n}{3}.$$

We rewrite the previous expression by replacing each k by its value in terms of the integers $0 \le q_k < n/m$ and $0 \le r_k < m/2$, where $k+1 = \frac{m}{2}q_k + r_k$.

$$cr(P) = \sum_{k=0}^{n/2-2}(n-mq_k-2r_k-1)\left(4\binom{\frac{m}{2}q_k+r_k+1}{2} - 4\binom{r_k+1}{2} + mr_k\right) - \frac{3}{4}\binom{n}{3}$$

$$= \sum_{q=0}^{\frac{n}{m}-1}\sum_{r=0}^{\frac{m}{2}-1}(n-mq-2r-1)\left(4\binom{\frac{m}{2}q+r+1}{2} - 4\binom{r+1}{2} + mr\right) - \frac{3}{4}\binom{n}{3}$$

$$= \frac{n}{48}\left(n^3 - 6n^2 + m^2n - 6mn + 22n - 12\right) = \frac{1}{2}\binom{n}{4} + \frac{1}{2}\binom{n/2}{2} + \frac{n^2}{6}\binom{m/2-1}{2}.$$

3.1 Construction

For an even integer m, and an arbitrary positive integer n, we construct a set A_n with mn points that achieves the minimum number of $\le k$-edges for every k among m-fold symmetric sets; and consequently it also has the minimum number of crossings among all m-fold symmetric sets.

Theorem 7 *Let $m > 0$ be even. For every positive integer n, there is a set A_n with mn points such that*

1. *$E_{\le k}(A_n) = 4(\binom{k+2}{2} - \binom{r+1}{2}) + mr$, for every $0 \le k \le mn/2-2$, where $k+1 = r \pmod{m/2}$.*
2. *$\text{cr}(A_n) = \frac{1}{2}\binom{mn}{4} + \frac{1}{2}\binom{mn/2}{2} + \frac{1}{6}\binom{m/2-1}{2}(mn)^2$.*

Here is the construction. Let $d = 2/\sin(\pi/(mn))$. We define the m-fold symmetric set A_n as follows

$$A_n = \left\{\left(d^j\cos\left(\frac{\pi(2an-j)}{mn}\right), d^j\sin\left(\frac{\pi(2an-j}{mn}\right)\right) : 0 \le a \le m-1, 0 \le j \le n-1\right\}.$$

For every $0 \le a \le m-1$ and $0 \le j \le n-1$ we use the notation

$$p(j,a) = \left(d^j\cos\left(\frac{\pi(2an-j)}{mn}\right), d^j\sin\left(\frac{\pi(2an-j}{mn}\right)\right).$$

Lemma 8 *If $0 \le j_1 \le j_2 < n-1$, $0 \le a_1 \le a_2 \le m-1$, and $(j_1,a_1) \ne (j_2,a_2)$, then the line through $p(j_1,a_1)$ and $p(j_2,a_2)$ is a halving line of the outermost m-gon in A_n; that is, it separates the set $\{p(a,n-1) : 0 \le a \le m-1\}$ into two parts each with $m/2$ points.*

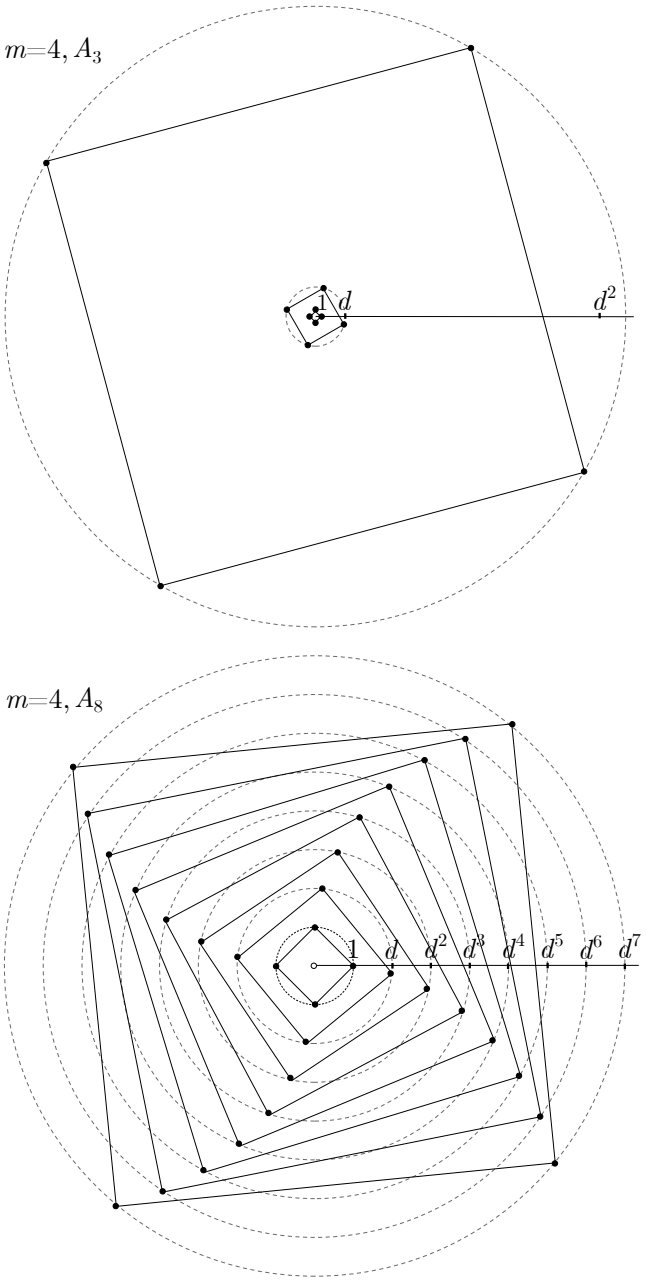


Figure 1: The 4-fold symmetric ($m = 4$) sets A_3 and A_8 described in the proof of Theorem 7. Since any more than 3 layers would be virtually impossible to see, the set A_8 is shown in logarithmic-polar scale to appreciate the rotations from one layer to the next.

Proof. (Sketch.) By rotational symmetry, assume that $a_1 = 0$ and for simplicity let $a_2 = a$. Furthermore, by using a dilation by a factor of d^{-j_1}, we can assume that $j_1 = 0$. Again by simplicity, we let $j_2 = j$, and we note that now we need to prove that the line through $p(0,0)$ and $p(j,a)$ halves the $c = n - j_1$ outer m-gon; that is, it halves the set $\{p(c,b) : 0 \le b \le m-1\}$. Note that $0 \le j \le c-1$, $0 \le a \le m-1$, and $(j,a) \neq (0,0)$. We will prove this assertion by showing that the points $p(c,b)$ and $p(c,b+m/2)$ lie on different sides of the line. To do this, we show that the product of the determinants

$$D = \begin{vmatrix} p(0,0) & 1 \\ p(j,a) & 1 \\ p(c,b) & 1 \end{vmatrix} \quad \text{and} \quad D' = \begin{vmatrix} p(0,0) & 1 \\ p(j,a) & 1 \\ p(c,b+m/2) & 1 \end{vmatrix}.$$

is negative (we omit the computations), and thus $p(c,b)$ and $p(c,b+m/2)$ lie on different sides of the line through $p(0,0)$ and $p(j,a)$. □

Lemma 9 *For every $1 \le a \le m/2-1$ the edge $p(n-1,0)p(n-1,a)$ is an $(a-1)$-edge of A_n.*

Proof. Suppose that $1 \le a \le m/2-1$. It is enough to show that the ray $p(n-1,0)p(n-1,a)$ leaves the set of points $\{p(n-1,b) : 1 \le b \le a-1\}$ on its right side, and the rest of points of A_n on its left side. Because the set $\{p(n-1,b) : 0 \le b \le n-1\}$ is a regular m-gon, then among these points only those with $1 \le b \le a-1$ are on the right side of $p(n-1,0)p(n-1,a)$. To finish the proof, we show that the rest of A_n, namely the set $A'_n = \{p(j,b) : 0 \le j \le n-2, 0 \le b \le m-1\}$ lies on the right side of $p(n-1,0)p(n-1,a)$. If $n = 1$, this is vacuously true, otherwise if R denotes the distance from the origin to the point $p(n-1,0)$, then the distance from the origin to the segment $p(n-1,0)p(n-1,a)$ is $R\cos(\pi a/m)$, and by construction A'_n is inside a circle of radius $R/d = R\sin(\pi/(mn))/2$ centered at the origin. Finally,

$$\begin{aligned} \cos\left(\frac{\pi a}{m}\right) &\ge \cos\left(\frac{\pi(m/2-1)}{m}\right) = \sin\left(\frac{\pi}{m}\right) \\ &= 2\sin\left(\frac{\pi}{2m}\right)\cos\left(\frac{\pi}{2m}\right) \ge 2\sin\left(\frac{\pi}{mn}\right)\cos\left(\frac{\pi}{4}\right) \\ &> 2\sin\left(\frac{\pi}{mn}\right)\cdot\frac{1}{4} = \frac{1}{2}\sin\left(\frac{\pi}{mn}\right), \end{aligned}$$

thus A'_n lies on the left side of the ray $p(n-1,0)p(n-1,a)$. □

Proof. [Proof of Theorem 7] We proceed to find an upper bound for the number of $\le k$-edges of A_n. By rotational symmetry consider the $\le k$-edges of the form $p(j,0)p(i,a)$ with $0 \le i \le j \le n-1$, $0 \le a \le m-1$, and $(j,0) \neq (i,a)$.

First suppose that $k \le m/2-2$. If $j = n-1$, then according to Lemma 9 the edges from $p(n-1,0)$ to

$p(i,a)$ with $i = n-1$ and $1 \le a \le k+1$ are all $\le k$-edges. Moreover, if $i < n-1$ and a is arbitrary, then the line $p(n-1,0)p(i,a)$ separates the sets of points $\{p(n-1,b) : 1 \le b \le m/2-1\}$ and $\{p(n-1,b) : m/2+1 \le b \le m-1\}$. Thus the edge $p(n-1,0)p(i,a)$ is a $\ge (m/2-1)$-edge. Finally, if $j < n-1$ and $i < n-1$, then by Lemma 8 the edge $p(n-1,0)p(i,a)$ is a $\ge (m/2)$-edge. Hence in this case all of the $\le k$-edges have both vertices in the outer m-gon where $i = j = n-1$, and there are exactly $2(k+1)$ such edges of the form $p(j,0)p(i,a)$. Hence there are exactly $m(k+1) \le k$-edges in A_n. In other words, if $k \le m/2-2$, then

$$E_{\le k}(A_n) = m(k+1). \tag{3}$$

Second, suppose that $m/2-1 \le k \le mn/2-3$. By Lemma 9 all of the edges with vertices in the outer m-gon, except the main diagonals, are $\le (m/2-1)$-edges, and so they are all $\le k$-edges as well. There are exactly $\binom{m}{2} - m/2$ such edges. In addition, by considering a radial sweep from each of the convex hull vertices $p(n-1,a)$ of A_n, we see that each of them is the endpoint of exactly two j-edges for every $m/2 \le j \le k$, where the other endpoint is not a vertex of the outer m-gon. There are $2m(k-(m/2-1))$ such edges. Finally, by Lemma 8, every j-edge of A'_n is a $(j+m/2)$-edge of A_n. Hence if $m/2-1 \le k \le mn/2-3$, then

$$\begin{aligned} &E_{\le k}(A_n) \\ &= E_{\le (k-m/2)}(A'_n) + \tbinom{m}{2} - \frac{m}{2} + 2m\left(k - \left(\frac{m}{2}-1\right)\right) \\ &= E_{\le (k-m/2)}(A'_n) + 2m(k+1) - m\left(\frac{m}{2}-1\right). \end{aligned} \tag{4}$$

From (3) and (4), we conclude that

$$\begin{aligned} E_{\le k}(A_n) = E_{\le k-m/2}(A'_n) + 2m(k+1) \\ - m \cdot \min(k+1, m/2-1). \end{aligned}$$

Therefore the set A_n achieves equality in Lemma 5. Following the proof of Theorem 6 we conclude that A_n achieves equality throughout as well. This assertion proves the first part, and the second follows because again the set A_n achieves equality every time Theorem 6 is used in the proof of Theorem 2. □

4 Odd symmetry

4.1 Bound improvement on k-edges

In this section, we use allowable sequences to improve Theorem 6 when m is odd. There, the lower bound counts $\le k$-transpositions of two types: those involving at least one of the m points that are removed during the induction and those that are $(\le k-(m+1)/2)$-transpositions in P'. However, there also might be $(k-(m-1)/2)$-transpositions in P' that are k-transpositions in P, and such $\le k$-transpositions have not been counted by the lower bound in Theorem 6. We show that when m is odd and k is large enough, such transpositions always exist. More precisely,

Lemma 10 *If m is odd and $\frac{m-1}{2} \cdot \frac{n}{m} \le k \le \lceil \frac{n}{2} \rceil - 2$, then there are at least $m(k+1) - \frac{m-1}{2}n$ different k-transpositions of P that are $(k-\frac{m-1}{2})$-transpositions of P'.*

Proof. We can assume that the convex hull ∂P consists of m points (one rotational orbit) and that the rest of the points of P are in the center of the star formed by the halving lines of ∂P. Moreover, we can assume that the rest of the points are as close as needed to the center of P so that all transpositions of any element of ∂P with the elements of $P - \partial P$ occurred consecutively on the circular sequence of P. Let Π be a half-period that shows the points of ∂P in the first $(m+1)/2$ or last $(m-1)/2$ positions of row 1 and such that the first $n-m$ transpositions of Π are transpositions of $x_{(m+1)/2}$ with the points in $P - \partial P$. (See Figure 2.) Then there exist rows $1 = s_1 \le t_1 \le s_2 \le t_2, \ldots, \le s_m \le t_m$ such that each row between s_i and t_i are with the i^{th} element of ∂P along the cycle (star) starting with the vertices $(m+1)/2, (m+3)/2, (m-3)/2, (m+7)/2, \ldots, 1, m, 2, m-2, 4, m-4, \ldots, (m-1)/2$ if $m \bmod 4 = 1$ or $(m+1)/2, (m+3)/2, (m-3)/2, (m+7)/2, \ldots, m, 1, m-1, 3, m-3, \ldots, (m-1)/2$ if $m \bmod 4 = 3$.

Partition the columns of Π into 3 regions: the left region L formed by the first $k+1$ elements of each row, the central region C formed by the next $n-2(k+1)$ elements of each row, and the right region R formed by the last $k+1$ elements of each row. This partition of Π inherits a partition of row r into the sets $L(r)$, $C(r)$, and $R(r)$. For $1 \le i \le (m-1)/2$, let

$$\begin{aligned} U_i &= L(s_{2i-1}) \cap (C(t_{2i}-1) \cup R(t_{2i}-1)), \\ V_i &= V_{i-1} \cap R(t_{2i}), \text{ where } V_0 = R(1), \\ W_i &= V_i \cap (L(s_{2i+1}-1) \cup C(s_{2i+1}-1)), \text{ and } W_0 = \emptyset. \end{aligned}$$

Because an element of V_{i-1} that is not in V_i either belongs to W_{i-1} or must be replaced by one of the $n-2(k+1)$ elements of $C(t_{2i-1})$ or by an element of U_i, then

$$|V_i| \ge |V_{i-1}| - |W_{i-1}| - (n-2(k+1)) - |U_i|.$$

Since $|V_0| = k+1$,

$$\begin{aligned} |V_{(m-1)/2}| \ge &(k+1) - \frac{m-1}{2} \cdot (n-2(k+1)) \\ &- \sum_{i=1}^{(m-3)/2} (|U_i| + |W_i|) - |U_{(m-1)/2}|. \end{aligned}$$

Finally, note that each element of U_i is involved in a k-transposition from row j to $j+1$ for some $t_{2i-1} \le j < s_{2i}$

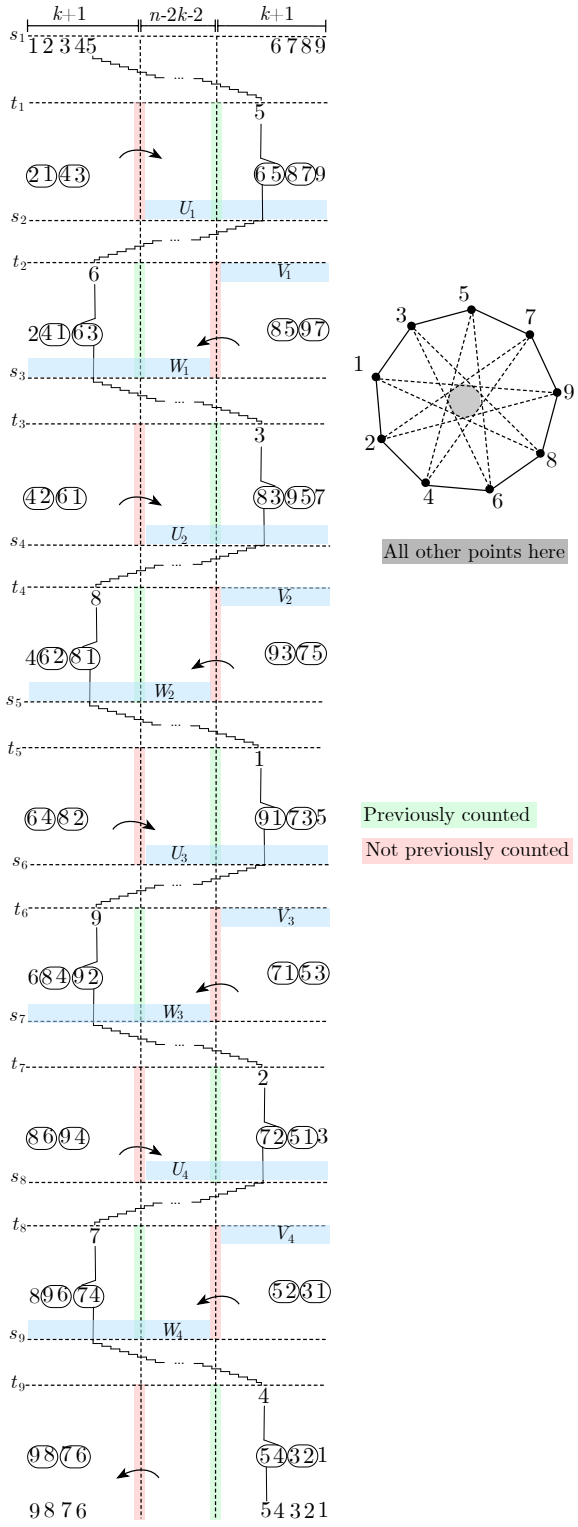


Figure 2: The structure of a 9-fold symmetric circular sequence.

and by taking the last such transposition (so that it involves exactly one element of U_i), we have a total of at least $|U_i|$ different k-transpositions in that range. Similarly, there are at least $|W_i|$ different k-transpositions from row j to $j+1$ for some $t_{2i} \le j < s_{2i+1}$. (Although this holds for $i = (m-1)/2$, we only use it for $1 \le i \le (m-3)/2$ in order to avoid double counting transpositions in the next argument.) Finally, any element of $V_{(m-1)/2}$ is involved in a k-transposition involving elements in positions $n-k-1$ and $n-k$ from some row j to $j+1$ for some $t_{m-1} \le j < s_m$ or in a k-transposition involving elements in positions $k+1$ and $k+2$ from some row j to $j+1$ for some $j \ge t_m$. Therefore, the number of desired k-transpositions is at least

$$|V_{(m-1)/2}| + \sum_{i=1}^{(m-3)/2} (|U_i| + |W_i|)| + |U_{(m-1)/2}|$$
$$\ge (k+1) - \frac{m-1}{2} \cdot (n - 2(k+1)) = m(k+1) - \frac{m-1}{2} n.$$

□

Our new lower bound is a direct application of Lemma 10.

Theorem 11 *Let $m \ge 3$ be odd. For any set P of n points with m-fold symmetry and for any $0 \le k \le \lceil n/2 \rceil - 2$,*

$$E_{\le k}(P) \ge 4 \binom{m}{m+1} \left(\binom{k+2}{2} - \binom{r+1}{2} \right) + mr + m \binom{k - \frac{m-1}{2} \cdot \frac{n}{m} + 2}{2},$$

where $k+1 = r \bmod \lceil m/2 \rceil$.

4.2 First crossing number lower bound improvement

We are now ready to present our first lower bound on the crossing number for odd symmetry, which follows from Theorems 1 and 11. In the next section, we present a further improvement that requires a slightly different approach. However, the next result is still needed for a large range of values of k.

Theorem 12 *For positive integers m and n such that m is odd and $m|n$, we have*

$$\text{sym-}\overline{\text{cr}}_m(K_n) \ge \left(\frac{m}{2(m+1)} + \frac{1}{8m^3} \right) \binom{n}{4} + \Theta(n^3).$$

Proof. Let $m \ge 3$ be odd an P be an m-fold symmetric set of n points in general position in the plane. By Theorem 2, we have

$$\text{Cr}(P) = \sum_{k=0}^{\lfloor n/2 \rfloor - 2} (n - 2k - 3) E_{\le k}(P) + \Theta(n^3)$$
$$= \binom{n}{4} \cdot 24 \sum_{k=0}^{\lfloor n/2 \rfloor - 2} \left(1 - \frac{2k}{n}\right) \frac{E_{\le k}(P)}{n^2} \cdot \frac{1}{n} + \Theta(n^3).$$

By Theorem 11, we have

$$\frac{E_{\leq k}(P)}{n^2} \geq \frac{4m}{m+1}\cdot\frac{k^2}{2n^2} + \frac{m}{2n^2}\cdot\max\left(0,\left(k-\frac{(m-1)n}{2m}\right)^2\right)+\Theta\left(\frac{1}{n}\right)$$
$$=\frac{2m}{m+1}\left(\frac{k}{n}\right)^2+\frac{m}{2}\left(\max\left(0,\frac{k}{n}-\frac{m-1}{2m}\right)\right)^2+\Theta\left(\frac{1}{n}\right).$$

Therefore,

$$\mathrm{Cr}(P) \geq \binom{n}{4}\int_0^{1/2} 24\,(1-2x)\left(\frac{2m}{m+1}x^2 + \frac{m}{2}\left(\max\left(0,x-\frac{m-1}{2m}\right)\right)^2\right)dx+\Theta(n^3)$$
$$=\binom{n}{4}\left(\frac{4m^4+m+1}{8m^3(m+1)}\right)+\Theta(n^3)$$
$$=\left(\frac{m}{2(m+1)}+\frac{1}{8m^3}\right)\binom{n}{4}+\Theta(n^3).$$

□

4.3 Second lower bound improvement: central approach

We conclude by noting that any improvements on the lower bound for $E_{\leq k}(P)$, automatically imply improvements on sym-$\overline{\mathrm{cr}}_m(K_n)$ by Theorem 2. For instance, using an approach similar to that in [4], we are able to improve the lower bound for $E_{\leq k}(P)$ for k close to $\lceil n/2\rceil - 2$ as follows:

Theorem 13 *Let $m \geq 3$ be odd. For any set P of n points with m-fold symmetry and for any $n/2 - n/(6m) \leq k \leq \lceil n/2\rceil - 2$,*

$$\frac{E_{\leq k}(P)}{n^2} \geq \frac{1}{2} - \frac{\sqrt{3m}(7m-1)}{9m(m+1)}\cdot\sqrt{1-\frac{2k}{n}}+\Theta\left(\frac{1}{n}\right).$$

Together with Theorem 2, this result improves Theorem 12 as shown below.

Theorem 14 *For positive integers m and n such that m is odd and $m|n$, we have for* sym-$\overline{\mathrm{cr}}_m(K_n)$ *as shown below.*

$$\text{sym-}\overline{\mathrm{cr}}_m(K_n) \geq \left(\frac{m}{2(m+1)}+\frac{1}{8m^3}+\frac{17m-11}{1080m^3(m+1)}\right)\binom{n}{4}+\Theta(n^3).$$

References

[1] B. M. Ábrego, O. Aichholzer, S. Fernández-Merchant, J. Leaños, and G. Salazar, There is a unique crossing-minimal rectilinear drawing of K_{18}, *Ars Mathematica Contemporanea*, 24(2):2–10, 2024.

[2] B. M. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and G. Salazar. The 2-page crossing number of K_n. *D*iscrete Comput. Geom., 49(4):747–777, 2013.

[3] B.M. Ábrego, O. Aichholzer, S. Fernńdez-Merchant, P. Ramos, and G. Salazar. Shellable drawings and the cylindrical crossing number of K_n. *Discrete Comput. Geom.*, 52:743–753, 2014.

[4] B. M. Ábrego, M. Cetina, S. Fernández-Merchant,J. Leaños, and G. Salazar. On $\leq k$-edges, crossings, and halving lines of geometric drawings of K_n. *D*iscrete Comput. Geom., 48(1):192–215, 2012.

[5] B. M. Ábrego, J. Dandurand, and S. Fernández-Merchant. The crossing number of centrally symmetric complete geometric graphs. *Procedia Comput. Sci.*, 195:275–279, 2021.

[6] B. M. Ábrego and S. Fernández-Merchant. A lower bound for the rectilinear crossing number. *Graphs and Combinatorics*, 21(3):293–300, 2005.

[7] B. M. Ábrego and S. Fernández-Merchant. Geometric drawings of K_n with few crossings. *J. Combin. Theory Ser. A*, 114:373–379, 2007.

[8] B. M. Ábrego, S. Fernández-Merchant, J. Leaños, and G. Salazar, A central approach to bound the number of crossings in a generalized configuration, *Electron. Notes Discrete Math.*, 30:273–278, 2008.

[9] O. Aichholzer. Another small but long step for crossing numbers: $cr(13) = 225$ and $cr(14) = 315$. *Proceedings of the 33rd Canadian Conference on Computational Geometry*, 72–77, 2021.

[10] O. Aichholzer, F. Aurenhammer, and H. Krasser. On the crossing number of complete graphs. *Computing*, 76:165–176, 2006.

[11] O. Aichholzer, F. Duque, R. Fabila-Monroy, C. Hidalgo-Toscano, and O. E. García-Quintero. An ongoing project to improve the rectilinear and the pseudolinear crossing constants. *J. Graph Algorithms and Applications*, 24(3):421–432, 2020.

[12] O. Aichholzer, J. García, D. Orden, and P. Ramos, New lower bounds for the number of ($\leq k$)-edges and the rectilinear crossing number of K_n. *D*iscrete Comput. Geom., 38:1–14, 2007.

[13] L. Beineke and R. Wilson, The early history of the brick factory problem, *Math. Intelligencer*, 32:41–48, 2010.

[14] F. Harary and A. Hill. On the number of crossings in a complete graph. *Proc. Edinburgh Math. Soc.*, 13:333–338, 1963.

[15] L. Lovász, K. Vesztergombi, U. Wagner, and E. Welzl. Convex quadrilaterals on k-sets. In *Towards a theory of geometric graphs*, 342 of *Contemp. Math.*, 139-148. Amer. Math. Soc., Providence RI, 2004.

[16] P. Turán. A note of welcome. *J. Graph Theory*, 1:7–9, 1977.

Local Fréchet Permutation

Jonathan James Perry* Benjamin Raichel*

Abstract

In this paper we consider computing the Fréchet distance between two curves where we are allowed to locally permute the vertices. Specifically, we limit each vertex to move at most k positions from where it started, and give fixed parameter tractable algorithms in this parameter k, whose running times match the standard Fréchet distance computation running time when k is a constant. Furthermore we also show that computing such a local permutation Fréchet distance is NP-hard[1] when considering the weak Fréchet distance.

1 Introduction

A polygonal curve in $\mathbb{R}^d$ is defined by linearly interpolating an ordered sequence of points. In this paper we consider the well studied topic of polygonal curve similarity, as measured by the standard Fréchet distance, but where we are allowed to permute the ordered sequence of vertices defining each curve. Specifically, we seek to determine if there are permutations such that the Fréchet distance of the resulting curves is at most δ, where $\delta > 0$ is some given distance parameter. Here we will limit each permutation to only allow for local reordering of the points. Namely, we consider *k-permutations*, where a point at position i before permutation must end up at a position j after permutation such that $|i - j| \leq k$. Limiting to the case when k is small, intuitively means the orderings before and after the permutation are close, and thus in some sense the resulting curves as well. In particular, it can model scenarios where the input curve has some local corruptions or where local faults in the ordering occurred when collecting the data. Critically, limiting k allows us to achieve efficient algorithms for the (strong) Fréchet distance, whereas conversely it appears to aid our proof of hardness for weak Fréchet.

Prior Work.
Several previous papers ([1, 2, 6, 9, 10]) have considered a variant of the Fréchet distance sometimes referred to as the Curve/Point Set Matching problem (CPSM). Here one is given a curve π (i.e. an ordered sequence of points) of length n and an unordered point set P of size m, and asked whether a subset of P can be selected and ordered such that its Fréchet distance to π is at most some given threshold δ. The results in these papers vary based on whether (i) discrete or continuous Fréchet distance is used, (ii) a proper subset or all points of P is used (subset vs. all-points), and (iii) points can be repeated or not (non-unique vs. unique).

We briefly review prior results on CPSM. [9] gave an $O(nm^2)$ time algorithm for the continuous, subset, non-unique variant. Subsequently, [10] considered the discrete version, showing NP-hardness for both the subset and all-points unique cases, and giving polynomial time algorithms for both the subset and all-points non-unique cases. The papers [1, 2] then filled out the remaining continuous cases showing that both unique and non-unique all-points as well as the unique subset variant are NP-hard. Finally, [6] showed that if both curves are point sets, then the all-points unique discrete Fréchet distance problem can be solved in $O((m+n)\log(mn))$ time, but is NP-hard in the continuous case.

Related to our motivation of having errant data, other methods have been proposed for fixing curves so as to minimize Fréchet distance. There are many prior works in this direction, though they are perhaps further away from our permutation problem than the CPSM problem. Here we mention only [5] and [7], as we will utilize their techniques as discussed below (for other related works see references in [5, 7]). [5] considered the strong and weak Fréchet distance where points on the curves are uncertain, meaning there is some given set of potential locations where the point may be realized. [7] considered the strong and weak Fréchet distance when deletions or insertions are allowed on one or both curves.

Our Results.
We introduce and study the k, ℓ-permutation Fréchet distance, where one is allowed to k-permute the first curve and ℓ-permute the second curve, with the goal of making the Fréchet distance of the resulting curves below some threshold δ. For both the continuous and discrete variants, we provide fixed parameter tractable algorithms in terms of k and ℓ. Our running times match the corresponding quadratic running times for the standard Fréchet distance algorithms when k and ℓ are constants. Namely, for curves of lengths n and m, and for both the discrete and continuous cases, we

*Department of Computer Science, University of Texas at Dallas, USA {jperry,benjamin.raichel}@utdallas.edu.
Work on this paper was partially supported by NSF CAREER Award 1750780 and CCF Award 2311179.

[1] As it is trivially in NP, NP-hardness implies NP-completeness, though for consistency throughout we use NP-hardness.

give a running time of $O(nmk^2 4^k)$ when one curve is permuted and running time of $O(nmk^2 4^k \ell^2 4^\ell)$ if both curves are permuted.

Observe that the unrestricted permutation case, i.e. when $k = n$ and $\ell = m$, has already been studied as this is equivalent to the all-points unique CPSM problem discussed above. The prior results on unrestricted permutations can then be compared to our results on restricted permutations, as seen in Table 1. This connection to prior work implies our restriction to k-permutations is required in order to achieve polynomial time algorithms for many of the cases. Furthermore, for our restricted setting, if we set $k = \ell = 0$, then our problem is equivalent to the decision version of computing the Fréchet distance, for which [3] proved that for both the discrete or continuous version there is no strongly subquadratic algorithm unless the Strong Exponential Time Hypothesis (SETH) fails. (In fact it was shown that assuming SETH no constant factor approximation exists, and the constant was subsequently improved in [4].) Thus our $O(mn)$ running time (i.e. $O(n^2)$ when $n = m$) for constant k and ℓ is essentially tight up to lower order factors assuming SETH. In particular, while [6] achieve a near linear running time for the special case when unrestricted permutations are allowed on both curves for discrete Fréchet distance, such a result is not possible in our restricted permutation setting assuming SETH.

Unrestricted Prior Results

	One Curve	Both Curves
Discrete	NP-hard [10]	$O((m+n)\log(mn))$ [6]
Continuous	NP-hard [2]	NP-hard [6]

Our Restricted Results : Theorem 5

	One Curve	Both Curves
Discrete	$O(nmk^2 4^k)$	$O(nmk^2 4^k \ell^2 4^\ell)$
Continuous	$O(nmk^2 4^k)$	$O(nmk^2 4^k \ell^2 4^\ell)$

Table 1: Comparison of our restricted k, ℓ-permutation results with prior unrestricted permutation results. The one curve results are obtained by dropping ℓ terms.

Additionally, we provide algorithmic results for these problems when the goal is minimization rather than decision, either when minimizing k when δ is fixed, or minimizing δ when k is fixed. Specifically, to find the minimal k, which we call κ, for both continuous and discrete Fréchet distance, we give a running time of $O(nm\kappa^2 4^\kappa)$ when one curve is permuted and $O(nm\kappa^4 16^\kappa)$ when both curves are permuted (in which case we require both curves to be restricted to κ-permutations). When minimizing δ for discrete Fréchet distance we provide a running time of $O(nmk^2 4^k)$ for one curve and $O(nmk^2 4^k \ell^2 4^\ell)$ for two curves. When minimizing δ for continuous Fréchet distance, let $n = \max\{|\pi|, |\sigma|\}$ and let $\tau = \max\{k, \ell\}$. We then provide a running time of $O(n^2 \tau^4 16^\tau (\log n + \tau))$ regardless of whether one or both curves are permuted, where the time bound holds with probability at least $1 - 1/n^c$, for any constant $c > 0$. These results are summarized in Table 2.

Min-K : Corollary 6

	One Curve	Both Curves
Disc./Cont.	$O(nm\kappa^2 4^\kappa)$	$O(nm\kappa^4 16^\kappa)$

Min-δ : Corollary 9 (Disc.) : Corollary 8 (Cont.)

	One Curve	Both Curves
Disc.	$O(nmk^2 4^k)$	$O(nmk^2 4^k \ell^2 4^\ell)$
Cont.	$O(n^2 \tau^4 16^\tau (\log n + \tau))$	$O(n^2 \tau^4 16^\tau (\log n + \tau))$

Table 2: Minimization results, where κ is the minimum k, $n = \max\{|\pi|, |\sigma|\}$, and $\tau = \max\{k, \ell\}$.

Finally, we consider the four decision problems but using weak Fréchet distance, which allows one to backtrack while traversing the curves, unlike the standard strong Fréchet considered in our algorithmic results. For the weak discrete Fréchet distance we reduce from 3SAT to show that permuting one curve or both curves is NP-hard even in $\mathbb{R}^1$. This reduction carries over to weak continuous Fréchet distance when raised to $\mathbb{R}^2$. In comparison, the prior NP-hardness results in Table 1 are all done in $\mathbb{R}^2$.

Both our algorithmic and hardness results follow a similar approach to that used in [7] for Fréchet edit distance. Specifically, our algorithmic results solve the problem by modeling potential solutions using DAG complexes, introduce in [8] and further utilized and expanded in [7]. Our hardness results for the weak case are inspired by the reduction used in [5], which was also used in [7] to prove weak variants of the Fréchet edit distance problem are NP-hard.

2 Preliminaries

Throughout, given points $p, q \in \mathbb{R}^d$, $||p - q||$ denotes their Euclidean distance. Moreover, given two (closed) sets $P, Q \subseteq \mathbb{R}^d$, $||P - Q|| = \min_{p \in P, q \in Q} ||p - q||$ denotes their distance, where for a single point $x \in \mathbb{R}^d$ we write $||x - P|| = ||\{x\} - P||$. We use angled brackets to denote an ordered list $\langle x_1, \ldots, x_n \rangle$, and use $L_1 \circ L_2$ to denote the concatenation of ordered lists L_1 and L_2. We use $[n]$ to denote the set $\{1, \ldots, n\}$.

Fréchet Distance.
The following definitions are standard, but in particular here we state the definitions directly as given in [7]. A *polygonal curve* is a sequence of n points $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ where $\pi_i \in \mathbb{R}^d$ for all i. Such a sequence induces a continuous mapping from $[1, n]$ to $\mathbb{R}^d$, which we also denote by π, such that for any integer $1 \leq i < n$, the restriction of π to the interval $[i, i+1]$ is defined by $\pi(i + \alpha) = (1 - \alpha)\pi_i + \alpha\pi_{i+1}$ for any $\alpha \in [0, 1]$, i.e. a straight line segment. We will view π as both a discrete

point sequence and a continuous function interchangeably, and when it is clear from the context, we also may use π to denote the image $\pi([1,n])$. We use $\pi[i,j]$, for $i \leq j$, to denote the restriction of π to the interval $[i,j]$. Given a curve $\pi = \langle \pi_1, \ldots, \pi_n \rangle$, we write $|\pi| = n$ to denote its size.

A reparameterization for a curve π of length n is a continuous non-decreasing bijection $f : [0,1] \to [1,n]$ such that $f(0) = 1, f(1) = n$. Given reparameterizations f, g of an n length curve π and an m length curve σ, respectively, the *width* between f and g is defined as

$$width_{f,g}(\pi,\sigma) = \max_{\alpha \in [0,1]} ||\pi(f(\alpha)) - \sigma(g(\alpha))||.$$

The (standard, i.e. continuous and strong) *Fréchet distance* between π and σ is then

$$\mathsf{d}_{\mathcal{F}}(\pi,\sigma) = \inf_{f,g} width_{f,g}(\pi,\sigma).$$

where f, g range over all possible reparameterizations of π and σ. Informally, the Fréchet distance is often described as the shortest leash length needed for a man on one curve and a dog on the other to walk from their respective starting to ending points of the curves.

The *discrete Fréchet distance* is similar to the above defined Fréchet distance, except that we do not traverse the edges but rather discontinuously jump to adjacent vertices. Specifically, define a monotone correspondence as a sequence of index pairs $\langle (i_1,j_1), \ldots, (i_k,j_k) \rangle$ such that $(i_1,j_1) = (1,1)$, $(i_k,j_k) = (n,m)$, for any $1 \leq z \leq k$ we have $1 \leq i_z \leq n$ and $1 \leq j_z \leq m$, and for any $1 \leq z < k$ we have $(i_{z+1}, j_{z+1}) \in \{(i_z+1, j_z), (i_z, j_z+1), (i_z+1, j_z+1)\}$. Let C denote the set of all monotone correspondences, then the discrete Fréchet distance is $\mathsf{d}_{\mathcal{DF}}(\pi,\sigma) = \inf_{c \in C} \max_{(i,j) \in c} ||\pi_i - \sigma_j||$.

Both the Fréchet distance and the discrete Fréchet distance have a corresponding *weak* variant, which is defined analogously except that one is allowed to backtrack on the curves. Specifically, the *weak Fréchet distance*, denoted $\mathsf{d}^{\mathrm{w}}_{\mathcal{F}}(\pi,\sigma)$, is defined similarly to the standard Fréchet distance above, except that when defining the width f and g are no longer required to be non-decreasing bijections, but are still required to be continuous and have $f(0) = 1, g(0) = 1$ and $f(1) = n, g(1) = m$. Similarly, the *weak discrete Fréchet distance*, denoted $\mathsf{d}^{\mathrm{w}}_{\mathcal{DF}}(\pi,\sigma)$, is defined similarly to the discrete Fréchet distance above, except that we no longer require the correspondence to be monotone. Specifically, a (non-monotone) correspondence is a sequence of index pairs $\langle (i_1,j_1), \ldots, (i_k,j_k) \rangle$ such that $(i_1,j_1) = (1,1)$, $(i_k,j_k) = (n,m)$, for any $1 \leq z \leq k$ we have $1 \leq i_z \leq n$ and $1 \leq j_z \leq m$, and for any $1 \leq z < k$ we have $(i_{z+1}, j_{z+1}) \in \{(i_z \pm 1, j_z), (i_z, j_z \pm 1), (i_z \pm 1, j_z \pm 1)\}$.

Permutation Fréchet Distance.
Viewing a polygonal curve $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ as a sequence of points, a permutation of π is a bijection $f : [n] \to [n]$, which induces a new curve $f(\pi) = \langle \pi_{f(1)}, \ldots, \pi_{f(n)} \rangle$. Given a permutation f of π, we refer to f as a *k-permutation* if $|f(i) - i| \leq k$ for all $1 \leq i \leq n$, and we let $\mathcal{P}_k(\pi)$ denote the set of all k-permutations of π.

The *k, ℓ-permutation Fréchet distance* is then

$$\mathsf{d}^{k,\ell}_{\mathcal{PF}}(\pi,\sigma) = \min_{f \in \mathcal{P}_k(\pi), g \in \mathcal{P}_\ell(\sigma)} \mathsf{d}_{\mathcal{F}}(f(\pi), g(\sigma))$$

Observe that $\mathcal{P}_0(\pi)$ consists only of the identity function. Thus if we wish to consider the problem where permutations are only allowed on π then we write $\mathsf{d}^{k}_{\mathcal{PF}}(\pi,\sigma) = \mathsf{d}^{k,0}_{\mathcal{PF}}(\pi,\sigma) = \min_{f \in \mathcal{P}_k(\pi)} \mathsf{d}_{\mathcal{F}}(f(\pi),\sigma)$. Moreover, observe that $\mathsf{d}_{\mathcal{F}}(\pi,\sigma) = \mathsf{d}^{0,0}_{\mathcal{PF}}(\pi,\sigma)$.

We now define several problems based on the above:

- In the min-k permutation Fréchet distance problem, denoted MinK-PF, for a given $\delta > 0$ we seek the smallest value k such that $\mathsf{d}^{k,k}_{\mathcal{PF}}(\pi,\sigma) \leq \delta$.
- In the min-δ permutation Fréchet distance problem, denoted Minδ-PF, for given k and ℓ we seek the smallest value δ such that $\mathsf{d}^{k,\ell}_{\mathcal{PF}}(\pi,\sigma) \leq \delta$.
- For the MinK-PF and Minδ-PF problems if we instead ask if $\mathsf{d}^{k}_{\mathcal{PF}}(\pi,\sigma) = \mathsf{d}^{k,0}_{\mathcal{PF}}(\pi,\sigma) \leq \delta$, then we respectively refer to it as the *one-sided* MinK-PF or Minδ-PF problem.

All of the above definitions and problems immediately extended to the discrete, weak, or discrete weak Fréchet distance by replacing $\mathsf{d}_{\mathcal{F}}(f(\pi), g(\sigma))$ respectively with $\mathsf{d}_{\mathcal{DF}}(f(\pi), g(\sigma))$, $\mathsf{d}^{\mathrm{w}}_{\mathcal{F}}(f(\pi), g(\sigma))$, or $\mathsf{d}^{\mathrm{w}}_{\mathcal{DF}}(f(\pi), g(\sigma))$, in the definition of k, ℓ-permutation Fréchet distance.

DAG Complexes.
We will utilize the work of [8] and [7], the first of which defines the following generalization of a curve. Consider a directed acyclic graph (DAG) with vertices in $\mathbb{R}^d$, where a directed edge $\mathsf{p} \to \mathsf{q}$ is realized by the directed segment pq. We refer to such an embedded graph as being a *DAG complex*, denoted $\mathcal{C}$, with embedded vertices $V(\mathcal{C})$ (i.e. points) and embedded edges $E(\mathcal{C})$ (i.e. line segments). We denote the size of the complex as $|\mathcal{C}| = |E(\mathcal{C})| + |V(\mathcal{C})|$. Note that a DAG complex is allowed to have crossing edges and overlapping vertices. Call a polygonal curve $\pi = \langle \pi_1, \ldots, \pi_k \rangle$ *compliant* with $\mathcal{C}$ if $\pi_i \in V(\mathcal{C})$ for all i and $\pi_i \pi_{i+1} \in E(\mathcal{C})$ for all $1 \leq i < k$. (Note this implies π traverses each edge in the direction compliant with its orientation from the DAG.)

The following theorem was given in [7], who observed that the original theorem from [8] easily generalizes to the case where one allows sets of points for the start and end rather than individual points.

Theorem 1 *Given two DAG complexes $\mathcal{C}_1$ and $\mathcal{C}_2$, initial vertices $S_1 \subseteq V(\mathcal{C}_1)$ and $S_2 \subseteq V(\mathcal{C}_2)$, target vertices $T_1 \subseteq V(\mathcal{C}_1)$ and $T_2 \subseteq V(\mathcal{C}_2)$, and a value δ, then in $O(|\mathcal{C}_1||\mathcal{C}_2|)$ time one can determine the set of all pairs $t_1 \in T_1$ and $t_2 \in T_2$, such that there are curves π_1 and π_2 such that*

- *π_i is compliant with $\mathcal{C}_i$ for $i = 1, 2$.*
- *π_i starts at some $s_i \in S_i$ and ends at t_i, for $i = 1, 2$.*
- *$\mathsf{d}_{\mathcal{F}}(\pi_1, \pi_2) \leq \delta$.*

The standard Fréchet distance decision problem is typically computed by considering the product complex of two curves (i.e. a grid), and propagating the the reachable space according to a topological ordering of the cells in this product. The above theorem is thus obtained by observing that when the input consists of DAG complexes (which generalize curves), then the same approach works as the product complex still consists of cells with a topological order.

For the discrete Fréchet distance between two curves, one can again propagate reachability through the product, except the product is no longer a continuous space but rather simply a discrete grid graph. Thus again we can generalize to the case when the input is a pair of DAG's. Specifically, given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, their product is $G_1 \times G_2 = (V_1 \times V_2, F)$, where $(u_1, u_2) \to (w_1, w_2) \in F$ if and only if (i) $u_1 = w_1$ and $u_2 \to w_2 \in E_2$, (ii) $u_2 = w_2$ and $u_1 \to w_1 \in E_1$, or (iii) $u_1 \to w_1 \in E_1$ and $u_2 \to w_2 \in E_2$. Observe, that since $G_1 \times G_2$ is also a DAG, there is thus again a topological ordering and so we immediately have the following corollary.

Corollary 2 *Given DAG's $G_1 = (V_1, E_2)$ and $G_2 = (V_2, E_2)$, with vertices in $\mathbb{R}^d$, initial vertices $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$, target vertices $T_1 \subseteq V_1$ and $T_2 \subseteq V_2$, and a value δ, then in $O(|G_1||G_2|)$ time one can determine the set of all pairs $t_1 \in T_1$ and $t_2 \in T_2$, such that there are paths π_1 in G_1 and π_2 in G_2 such that*

- *π_i starts at some $s_i \in S_i$ and ends at t_i, for $i = 1, 2$.*
- *$\mathsf{d}_{\mathcal{DF}}(\pi_1, \pi_2) \leq \delta$.*

3 Permutation Fréchet Distance

Given a polygonal curve $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ and a parameter k, our goal is to construct a DAG complex where the set of compliant paths between specified start and end vertices is the same as the set of all k-permutations of π. To this end, consider constructing an arbitrary k-permutation of π, denoted $f(\pi)$, one vertex at a time. Observe that since f is a k-permutation, there are at most $2k + 1$ possible candidates for the vertex $\pi_{f(i)}$ for any i, namely the set $\{\pi_j \mid |j - i| \leq k\}$. Viewing these candidates as an ordered set $\langle \pi_{i-k}, \ldots, \pi_{i+k} \rangle$, any subset can be represented by a binary vector $v = \langle v_1, \ldots, v_{2k+1} \rangle \in \{0, 1\}^{2k+1}$, where $v_j = 1$ represents that $\pi_{i-k+(j-1)}$ is in the subset and $v_j = 0$ represent that it is not. In particular, when considering the possibilities for $\pi_{f(i)}$ we wish to restrict to the subset of $\langle \pi_{i-k}, \ldots, \pi_{i+k} \rangle$ which did not occur already in $\langle \pi_{f(1)}, \ldots, \pi_{f(i-1)} \rangle$. Thus rather than remembering this entire prior sequence, it suffices to pass a single $2k + 1$ length binary vector representing the subset of $\langle \pi_{i-k}, \ldots, \pi_{i+k} \rangle$ that has occurred already.

The construction of our DAG complex $\mathcal{C}$ is thus as follows. The vertices of $\mathcal{C}$ are copies of the vertices from π. Specifically, for each original vertex π_i we create a copy $\pi_i^{j,v}$ (i.e. π_i and $\pi_i^{j,v}$ have the same location) where j represents that π_i is the jth vertex in the permutation f, and $v \in \{0, 1\}^{2k+1}$ represents the subset of the $(2k + 1)$ possible vertices for $\pi_{f(j+1)}$ that have already occurred, as described above. Thus we have $V(\mathcal{C}) = \{\pi_i^{j,v} \mid 1 \leq i \leq n, |j - i| \leq k, v \in \{0, 1\}^{2k+1}\}$, and observe that $|V(\mathcal{C})| = O(nk2^{2k}) = O(nk4^k)$.

For the edge set $E(\mathcal{C})$, consider some vertex $\pi_i^{j,v}$. We add an edge from $\pi_i^{j,v}$ to $\pi_z^{j+1,w}$ if and only if $|j + 1 - z| \leq k$, π_z did not occur already (i.e. was not represented by a 1 in v), and w is consistent with v. In order for w to be consistent with v, w must represent the subset of the $(2k + 1)$ possible vertices for $\pi_{f(j+2)}$ that have already occurred, given that v represented the subset of the $(2k + 1)$ possible vertices for $\pi_{f(j+1)}$ that have already occurred. Thus $w = \langle w_1, \ldots, w_{2k+1} \rangle$ being consistent with $v = \langle v_1, \ldots, v_{2k+1} \rangle$ means that $w_{2k+1} = 0$ and $w_i = v_{i+1}$ for all $i < 2k + 1$, with the exception that the entry in w for π_z must be set to 1. Thus the degree of vertex $\pi_i^{j,v}$ is $O(k)$ and so $|\mathcal{C}| = O(|V(\mathcal{C})| + |E(\mathcal{C})|) = O(k \cdot nk4^k) = O(nk^2 4^k)$.

The last thing we must define is the allowed starting vertices S and ending vertices T in the DAG complex. Naturally, S consists of all vertices of the form $\pi_i^{1,v}$ where v is the all 0 vector except for the position representing π_i being 1, as such a vertex represents that π_i will be the first vertex in the permutation and it thus is the only vertex excluded from possibilities for $\pi_{f(2)}$. Similarly, T consists of all vertices of the form $\pi_i^{n,v}$, where here we can allow v to be any binary vector (since if the bit setting is invalid, it will not be reachable from a vertex in S).

As the above described complex $\mathcal{C}$ can be constructed in linear time in its size, we have the following.

Lemma 3 *Given a polygonal curve $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ and a parameter k, in $O(nk^2 4^k)$ time one can construct a DAG complex $\mathcal{C}$ of size $O(nk^2 4^k)$ with vertex subsets S and T, such that the set of compliant paths in $\mathcal{C}$ that start at a vertex in S and end at a vertex in T is the same as the set of all k-permutations of π.*

It is easy to see that the above immediately applies to discrete Fréchet distance, by simply constructing the

corresponding DAG's for the curves, rather than the DAG complexes. Thus we have the following.

Corollary 4 *Given a curve $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ and a parameter k, in $O(nk^2 4^k)$ time one can construct a DAG G of size $O(nk^2 4^k)$ with vertex subsets S and T, such that the set of paths in G that start at a vertex in S and end at a vertex in T is the same as the set of all k-permutations of π.*

3.1 Algorithms and Results

Given curves π and σ of lengths n and m, respectively, along with integer parameters k and ℓ, and a value $\delta > 0$, Lemma 3 can be used to determine if $\mathsf{d}^{k,\ell}_{\mathcal{PF}}(\pi, \sigma) \leq \delta$. First, using Lemma 3, for the curve π and the parameter k we build a complex $\mathcal{C}_\pi$ along with sets S_π and T_π. Similarly for the curve σ and the parameter ℓ we build a complex $\mathcal{C}_\sigma$ with sets S_σ and T_σ. By definition, $\mathsf{d}^{k,\ell}_{\mathcal{PF}}(\pi, \sigma) \leq \delta$ if there exists $f \in \mathcal{P}_k(\pi)$ and $g \in \mathcal{P}_\ell(\sigma)$ such that $\mathsf{d}_{\mathcal{F}}(f(\pi), g(\sigma)) \leq \delta$, which by Lemma 3 is true if and only if there is a compliant path in $\mathcal{C}_\pi$ that starts at a vertex in S_π and ends at a vertex in T_π along with a compliant path in $\mathcal{C}_\sigma$ that starts at a vertex in S_σ and ends at a vertex in T_σ, such that their Fréchet distance is at most δ. Thus by applying Theorem 1 we have the following. Note by instead using Corollary 4 and Corollary 2 we can also handle the discrete case.

Theorem 5 *Given curves π and σ of lengths n and m, respectively, along with integers k and ℓ, and a value $\delta > 0$, in $O(nmk^2 4^k \ell^2 4^\ell)$ time one can determine if $\mathsf{d}^{k,\ell}_{\mathcal{PF}}(\pi, \sigma) \leq \delta$, for either the discrete or continuous Fréchet distance. The one sided problem can be solved in $O(nmk^2 4^k)$ by setting if ℓ to 0.*

Recall the MinK-PF problem defined in Section 2, where our goal is to find the minimum value k such that $\mathsf{d}^{k,k}_{\mathcal{PF}}(\pi, \sigma) \leq \delta$. Note that k cannot exceed $\max\{n, m\}$. Thus using Theorem 5, we search for the smallest k such that $\mathsf{d}^{k,k}_{\mathcal{PF}}(\pi, \sigma) \leq \delta$. Observe that the running time in the theorem is exponential in k, and in particular the running time for $k+1$ is a constant factor larger than that for k. Thus if we search for the minimum value, by successively incrementing k by 1, the times of the calls will behave like an increasing geometric series, and thus the overall time will be proportional to the last call.

Corollary 6 *Let κ be the optimal value for the MinK-PF problem. Then the MinK-PF problem can be solved in $O(nm\kappa^4 16^\kappa)$ time, for either the discrete or continuous Fréchet distance. The one-side MinK-PF problem can be solved in $O(nm\kappa^2 4^\kappa)$ time.*

[8] showed how to turn the decision procedure of Theorem 1 into an optimization procedure, using a simple sampling based approach which avoids the more complicated parametric search technique typically used to compute the Fréchet distance. The algorithm is guaranteed to be correct, and while its running time is a random variable (i.e. it is a Las Vegas algorithm), with polynomially high probably it achieves an efficient running time.

We remark that technically the following theorem from [8] was originally stated with only a single start and end vertex in each complex, whereas we require allowing sets of start and end vertices, although it immediately generalizes to this case. Specifically, [8] searches over a set of critical values using a decision procedure. The decision procedure was already generalized in [7] to sets of start and end vertices, as stated above in Theorem 1. Moreover, the critical values remain exactly the same when generalizing to starting and ending vertex sets. (This holds as [8] considered all vertex to vertex pairs as critical events, not simply just the pair of starting vertices and pair of ending vertices.) Thus we have the following.

Theorem 7 ([8], Theorem 6.3) *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two DAG complexes, of total complexity n, with start and end vertex sets $S_1, T_1 \subseteq V(\mathcal{C}_1)$, $S_2, T_2 \subseteq V(\mathcal{C}_2)$. Then there is an algorithm which computes two curves π_1 and π_2 such that π_1 (resp. π_2) is compliant with $\mathcal{C}_1$ (resp. $\mathcal{C}_2$), starts at a vertex in S_1 (resp. S_2), and ends at a vertex of T_1 (resp. T_2). Moreover, $\mathsf{d}_{\mathcal{F}}(\pi_1, \pi_2)$ is minimum among all such curves. The running time of the algorithm is $O(n^2 \log n)$ with probability at least $1 - 1/n^c$, for any constant $c > 0$.*

Corollary 8 *Let $n = \max\{|\pi|, |\sigma|\}$ and let $\tau = \max\{k, \ell\}$. Then both the one-sided and two-sided versions of the continuous Minδ-PF problem can be solved in $O(n^2 \tau^4 16^\tau (\log n + \tau))$ time, where the time bound holds with probability at least $1 - 1/n^c$, for any constant $c > 0$.*

It is well known that for the discrete Fréchet distance both the decision and optimization problem can be solved in $O(nm)$ time, since rather than propagating reachability in the free space we can propagate the minimum cost to reach the given vertex (which works as the free space is a discrete DAG). Again, in our case even though the input is now two DAG's rather than simply two curves, the product is still just a DAG. Thus we also can solve the optimization problem in the same time as our decision algorithm, yielding the following.

Corollary 9 *Given curves π and σ of lengths n and m, respectively, along with integers k and ℓ, one can solve the discrete Minδ-PF problem in $O(nmk^2 4^k \ell^2 4^\ell)$ time, which becomes $O(nmk^2 4^k)$ time for the one sided case.*

4 Hardness for Weak Permutation Fréchet

We now shift focus to *weak Fréchet distance* and prove NP-hardness for both the discrete and continuous cases $\mathsf{d}^{\mathrm{w}\ k,\ell}_{\mathcal{PDF}}(\pi,\sigma) \leq \delta$ in $\mathbb{R}^1$ and $\mathsf{d}^{\mathrm{w}\ k,\ell}_{\mathcal{PF}}(\pi,\sigma) \leq \delta$ in $\mathbb{R}^2$ for any constants $k \geq 1$ and $\ell \geq 0$. Our reductions are from 3SAT and closely follow those of [5] and [7]. We consider discrete curves in $\mathbb{R}^1$ first before moving to $\mathbb{R}^2$.

We set δ to 1 and rely on the resulting *Free Space Diagram* (ex. Figure 1), which is built by listing π-values as rows, σ-values as columns, and drawing empty circles where $|\pi_i - \sigma_j| \leq \delta$ (free spaces). We use teal to show paths between free spaces, with movement restricted to adjacent spaces (including diagonals). With this, $\mathsf{d}^{\mathrm{w}}_{\mathcal{DF}}(\pi,\sigma) \leq 1$ if and only if there is a way to traverse through the teal from the bottom left corner to the top right corner.

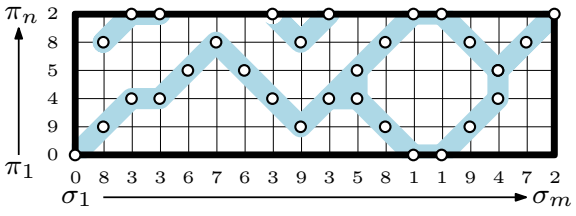


Figure 1: Generic Free Space Diagram.

We can now show the abstract Figure 2, where we embed the 3SAT instance with c clauses into σ with clause gadgets and walls such that $\sigma = \langle\text{wall}\rangle \circ \langle\text{clause 1}\rangle \circ \langle\text{wall}\rangle \circ \cdots \circ \langle\text{wall}\rangle \circ \langle\text{clause } c\rangle \circ \langle\text{wall}\rangle$. We construct π in three sections $\pi = \langle\text{lower}\rangle \circ \langle\text{variable layer}\rangle \circ \langle\text{upper}\rangle$. The interaction between the curves will force the length of π to be traversed for each clause, with 3 options representing satisfying one literal each. The clauses are placed in series, alternating going up or down, such that all clauses must have at least one literal satisfied in order to traverse to the end of σ.

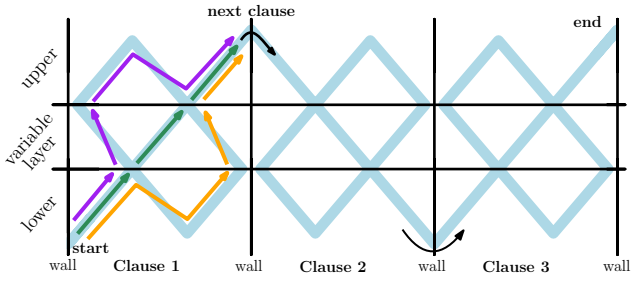


Figure 2: Abstract figure with sections of π and σ labeled. Purple, green, and orange illustrate the 3 ways to get through a clause.

The variable layer holds pairs of values where each pair represents a variable. Whether the pair is permuted or not determines if the variable is set to True or False. We call these permutations *sanctioned* (and all others *unsanctioned*) as these are the only ones that correspond to 3SAT actions. Further, our construction ensures only sanctioned permutations will be useful.

Precise Reduction.
Having seen the general idea, the following is the precise construction for π and σ given a 3SAT instance I with v variables and c clauses, an example of which is shown in Figure 3. Note that we will insert copies of a special point ρ and duplicate others, both of which serve only to discourage unsanctioned permutations.

Let $\rho = \infty$ in $\mathbb{R}^1$ for discrete and $\rho = (0,\infty)$ in $\mathbb{R}^2$ for continuous.[2] Note that when working in $\mathbb{R}^2$, all other points will be of the form $(x,0)$ for some value x. Additionally let $S^{[x]}$ be the concatenation of x instances of ordered set S (e.g. $\langle 5,1\rangle^{[2]} = \langle 5,1\rangle \circ \langle 5,1\rangle$). For any ordered set S we define S^R as that set in reverse order.

..... **construct** π .....

- Let $\odot$ represent '$\circ\ \langle\rho\rangle^{[2k+1]}\ \circ$'.
- Let $L = \langle 15\rangle \odot \langle 25\rangle \odot \ldots \odot \langle 10v+5\rangle$.
- Let $\hat{L}$ be L but with the value $10j+5$ replaced by $\langle 10j+4, 10j+6\rangle$ for all $1 \leq j \leq v$.
- Let $\pi = \langle 0\rangle \odot \langle 10\rangle \odot L \odot \langle 10(v+1)\rangle \odot \hat{L}^R \odot \langle 10\rangle \odot L \odot \langle 10(v+1)\rangle \odot \langle 10(v+2)\rangle$.

..... **construct** σ .....

- Let $\circledcirc$ represent '$\circ\ \langle\rho\rangle^{[2\ell+1]}\ \circ$'.
- Let $L = \langle 15\rangle \circledcirc \langle 25\rangle \circledcirc \ldots \circledcirc \langle 10v+5\rangle$.
- Let L^+_j be obtained from L by replacing the value $10j+5$ with $\langle 10j+4\rangle^{[\ell+1]} \circ \langle 10j+6\rangle^{[\ell+1]}$.
- Let L^-_j be obtained from L by replacing the value $10j+5$ with $\langle 10j+6\rangle^{[\ell+1]} \circ \langle 10j+4\rangle^{[\ell+1]}$.
- Let σ^i represent clause i of I which contains variables $\mathcal{X}_{j_1}$, $\mathcal{X}_{j_2}$, and $\mathcal{X}_{j_3}$ and therefore $\sigma^i = \langle 10\rangle \circledcirc L^{\pm}_{j_1} \circledcirc \langle 10(v+1)\rangle \circledcirc (L^{\pm}_{j_2})^R \circledcirc \langle 10\rangle \circledcirc L^{\pm}_{j_3} \circledcirc \langle 10(v+1)\rangle$, where $L^{\pm}_{j_i} = L^+_{j_i}$ if $\mathcal{X}_{j_i}$ appears as a positive literal and $L^{\pm}_{j_i} = L^-_{j_i}$ if $\mathcal{X}_{j_i}$ appears as a negated literal.
- Let $\sigma = \langle 0\rangle \circledcirc \sigma^1 \circledcirc \langle 10(v+2)\rangle \circledcirc (\sigma^2)^R \circledcirc \langle 0\rangle \circledcirc \ldots \circledcirc \sigma^c \circledcirc \langle 10(v+2)\rangle$ (if c is even, duplicate one clause so the total number of clauses is odd).

From this reduction comes the following two theorems which we spend the remainder of the paper proving.

Theorem 10 *Determining if* $\mathsf{d}^{\mathrm{w}\ k,\ell}_{\mathcal{PDF}}(\pi,\sigma) \leq \delta$ *in* $\mathbb{R}^1$ *for any constants* $k \geq 1$ *and* $\ell \geq 0$ *is NP-hard.*

[2] ∞ can be replaced with a sufficiently large finite value far away from the other values as explained in [7].

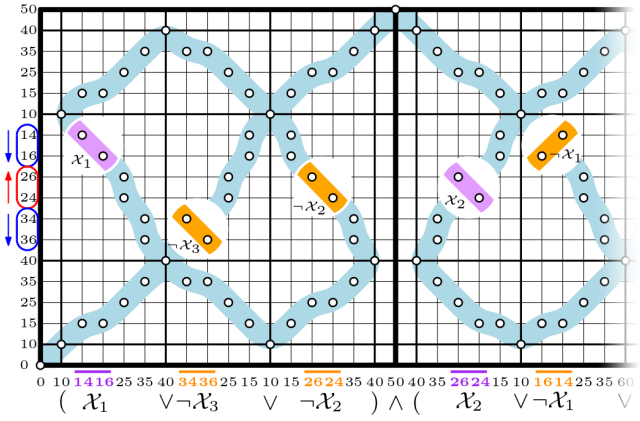


Figure 3: An example where $\mathcal{X}_2$ is set to False and other variables to True as seen in circled pairs on the left. ρ values are not shown, and $\ell = 0$. Figure faded out after second literal of second clause.

Theorem 11 *Determining if* $\mathsf{d}^{\mathrm{w}\ k,\ell}_{\mathcal{PF}}(\pi,\sigma) \le \delta$ *in* $\mathbb{R}^2$ *for any constants* $k \ge 1$ *and* $\ell \ge 0$ *is NP-hard .*

Proof of correctness for Theorem 10.
We now argue correctness. First, we argue that if only sanctioned permutations are made, the distance is $\le \delta$ if and only if I is satisfiable. Second, we argue that unsanctioned permutations are futile and will not result in the distance being $\le \delta$ unless it would have been possible without them.

So suppose that only sanctioned permutations are allowed. Consider the variable layer, where the jth variable $\mathcal{X}_j$ is represented by consecutive rows (i.e. a sanctioned permutation pair of vertices from π) with value $10j+6$ followed by value $10j+4$, which we can either leave in that order or permute (see Figure 3). We argue that leaving this pair unpermuted corresponds to setting the variable to True and permuting it corresponds to setting the variable to False. Specifically, consider the variable layer restricted to the columns of the ith clause, represented by σ^i. If $\mathcal{X}_j$ does not appear in this clause, then because the corresponding value $10j+5$ in σ^i is within distance $\delta = 1$ of both $10j+4$ and $10j+6$, the three paths through the variable layer (again see Figure 2) will be unobstructed at the rows corresponding to $\mathcal{X}_j$, regardless of whether we permute this sanctioned permutation pair or not.

Conversely, if $\mathcal{X}_j$ is in the ith clause then rather than $10j+5$ in σ^i we have the values $10j+4$ and $10j+6$ (possibly with repetition if $\ell > 0$). The portion of σ^i corresponding to $\mathcal{X}_j$ is either ascending or descending, depending on the parity of i and whether $\mathcal{X}_j$ is the first, second, or third variable in the clause. For simplicity, assume it is descending (the ascending case is symmetric). In this case, by construction, the pair of values $10j+4$ and $10j+6$ will appear in decreasing order if $\mathcal{X}_j$ occurs as a positive literal in clause i, and in increasing order if $\mathcal{X}_j$ occurs as a negated literal in clause i. So suppose $\mathcal{X}_j$ appears a positive literal, and consider the sanctioned permutation pair on π corresponding to $\mathcal{X}_j$. As the values in the variable layer are also descending (before any permutation occurs), in order to pass through the rows of this sanctioned permutation pair along the path corresponding to positive literal $\mathcal{X}_j$ in this clause, we *must not* permute the sanction pair (i.e. the rows and columns must both agree to descend in value for this pair). Conversely, if $\mathcal{X}_j$ appears as a negated literal then $10j+4$ and $10j+6$ will appear in increasing order on σ^i and so we *must* permute the sanctioned permutation pair to match if we want to pass along the corresponding path. As at least one of three paths through the variable layer for this clause must be passable, this corresponds to setting the variables in such a way that at least one of the literals in the clause is True, as required for the 3SAT instance I to be satisfiable. Therefore, if only sanctioned permutations are allowed, $\mathsf{d}^{\mathrm{w}\ k,\ell}_{\mathcal{PDF}}(\pi,\sigma) \le \delta$ if and only if I is satisfiable.

To complete the proof we now argue that even if unsanctioned permutations are allowed, they will not lower the Fréchet distance, and hence are futile. Specifically, by inserting a sufficiently large number of ρ points between adjacent pairs of non-ρ points, we insured that k,ℓ-permutations could not change the ordering of the those pairs. On π the only time ρ was not inserted was between points in pairs whose permutation was sanctioned. On σ the only time ρ was not inserted was between points in pairs that enforce a literal.[3] However, for each such pair $\langle x, y\rangle$ we duplicated x and y both $\ell+1$ times. This means regardless of any ℓ-permutation, the first copy of x must come before the first of y and the last copy of x before the last of y. This is sufficient since when the path of this literal is broken due to incorrect variable assignment, it is because the free space of both the first x and the last y become disconnected from the rest of the path.[4] If this disconnect exists before permutation, it will remain as the first x and last y remain. Finally, observe that a ρ point on one curve must map to a ρ point on the other curve, but the initial construction inserted ρ points between all pairs that were neither sanctioned on π nor representing a literal on σ, and we inserted enough such that there must remain at least one ρ between these pairs even after k,l-permutation. Therefore futility is established.

Proof of correctness for Theorem 11.
As was done in [5] and [7], we can extend the argument to the continuous Fréchet distance by raising to $\mathbb{R}^2$, us-

[3] Inserting ρ between these would have given π issues traversing its sanctioned permutation pairs

[4] Note that permutation does not change the free spaces, just their order. Thus we refer to incorrect assignment as 'disconnecting' the free spaces rather than making them 'no longer free'.

ing $\rho = (0, \infty)$, and converting $x \to (x, 0)$ for any other point x in the above reduction. We now argue this limits continuous to have Fréchet distance $\leq \delta$ if and only if discrete would have as well.

Consider a pair of traversals, one on each curve, such that their Fréchet distance is finite. Then on both curves, these traversals must simultaneously start at the first vertex, and go the next vertex (which is ρ) which is always possible in both continuous and discrete. The same is true for traversing between the second to last vertex (which is ρ) and the last on both curves. We break the remaining traversals of both curves into sub-traversals that start at a ρ, traverse some potentially different number of non-ρ vertex(es), and end at a ρ. Let Ξ_π (resp. Ξ_σ) be the non-ρ vertices traversed in an arbitrary one of these sub-traversals on π (resp. σ) where vertices may appear multiple times due to using weak Fréchet distance. Recall because the Fréchet distance is finite, that a ρ on the traversal of one curve must map to a ρ on the other. This means that a sub-traversal between ρ vertices on one curve must map to such a sub-traversal on the other, and one can argue that whether such sub-traversals are within Fréchet distance δ is equivalent to whether the corresponding Ξ_π and Ξ_σ are within Fréchet distance δ. (We can assume Ξ_π and Ξ_σ are non-empty, since if they were both empty we did not make any actual progress in traversing either curve. If Ξ_π was empty and $\Xi_\sigma = \langle x \dots \rangle$ then for the Fréchet distance to be $\leq \delta$, the sub-traversal of π must be approaching a vertex y where $||x - y|| \leq \delta$, and so one can argue y could have been included.)

The above implies it suffices to show that $\mathsf{d}_{\mathcal{DF}}(\Xi_\pi, \Xi_\sigma) \leq \delta \iff \mathsf{d}_{\mathcal{F}}(\Xi_\pi, \Xi_\sigma) \leq \delta$ for all Ξ_π and Ξ_σ. There are two types of Ξ_π: i) individual points, and ii) those including at least two vertices in a pair representing a variable assignment. Likewise there are two types of Ξ_σ: i) individual points, and ii) those including at least two vertices in a pair (duplicated if $\ell > 0$) which enforces a literal.

If either Ξ_π or Ξ_σ is a single point, then the discrete and continuous Fréchet distances are $\leq \delta$ iff. all points on the other curve are within δ of it. This shows the equivalence for all the cases except the distance between type ii of Ξ_π and type ii of Ξ_σ, so consider this case. Ξ_π cannot contain vertices from more than one variable, and the same is true for Ξ_σ w.r.t literals. Additionally, if they do not refer to the same variable/literal, or they refer to the same variable/literal but the variable is not set in such a way that the literal is satisfied, then the the Fréchet distance is $> \delta$ since the first vertices are too far from one another. Now observe that if both Ξ_π and Ξ_σ refer to the same variable/literal and the variable is set in such a way to satisfy the literal, then the Fréchet distance $\leq \delta$ if and only if they start and end with the same value, regardless of discrete or continuous.

References

[1] Paul Accisano and Alper Üngör. Approximate matching of curves to point sets. In *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG)*. Carleton University, Ottawa, Canada, 2014. URL: http://www.cccg.ca/proceedings/2014/papers/paper65.pdf.

[2] Paul Accisano and Alper Üngör. Hardness results on curve/point set matching with fréchet distance, 2012. arXiv:1211.2030.

[3] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, page 661–670, USA, 2014. IEEE Computer Society. doi:10.1109/FOCS.2014.76.

[4] Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *JoCG*, 7(2):46–76, 2016.

[5] Kevin Buchin, Maarten Löffler, Tim Ophelders, Aleksandr Popov, Jérôme Urhausen, and Kevin Verbeek. Computing the Fréchet distance between uncertain curves in one dimension. *Comput. Geom.*, 109:101923, 2023. doi:10.1016/j.comgeo.2022.101923.

[6] Maike Buchin and Bernhard Kilgus. Fréchet distance between two point sets. *Computational Geometry*, 102:101842, 2022. URL: https://www.sciencedirect.com/science/article/pii/S0925772121000985, doi:10.1016/j.comgeo.2021.101842.

[7] Emily Fox, Amir Nayyeri, Jonathan James Perry, and Benjamin Raichel. Fréchet edit distance, 2024. arXiv:2403.12878.

[8] Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, 2014. doi:10.1145/2532646.

[9] Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Staying close to a curve. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG)*, 2011. URL: http://www.cccg.ca/proceedings/2011/papers/paper97.pdf.

[10] Tim Wylie and Binhai Zhu. Following a curve with the discrete fréchet distance. *Theoretical Computer Science*, 556:34–44, 2014. Combinatorial Optimization and Applications. URL: https://www.sciencedirect.com/science/article/pii/S0304397514004629, doi:10.1016/j.tcs.2014.06.026.

PersiSort: A New Perspective on Adaptive Sorting Based on Persistence

Jens Kristian Refsgaard Schou* Bei Wang†

Abstract

Adaptive sorting exploits the structure of a partially sorted list—in particular, the sorted segments of a list called runs—to improve its performance. Persistent homology, on the other hand, is a topological data analysis tool that captures a space's topological features at different scales. In this paper, we combine these two seemingly unrelated concepts and introduce a new perspective on adaptive sorting. We introduce a new stable sorting algorithm, referred to as the Persistence Sort (or PersiSort in short), which utilizes the persistence pairs among the local extrema of a list. Given a list of n elements containing r runs with run entropy H, we prove, for the first time, that any adaptive sorting algorithm that uses the two-way-merge subroutine (AdaptMerge) of Carlsson et al. (1990) performs $\mathcal{O}(nH) = \mathcal{O}(n \log r)$ comparisons to merge precomputed runs based on its predetermined merge policy, and is therefore worst-case optimal. Using PersiSort, we then provide a new way to analyze adaptive sorting with a persistence-based arrangement of runs. Unlike previous work such as PowerSort and TimSort, PersiSort does not consider the number of elements in each run but the values of elements in the sorting process. We finally discuss the scenarios when PersiSort outperforms several state-of-the-art adaptive sorting algorithms.

1 Introduction

A sorting algorithm has a basic goal: putting elements from a list into some total order. Adaptive sorting is an active area of research that exploits the structure of a partially sorted list to improve performance. Specifically, it utilizes unique structures in the input called the *runs*, which are segments of the list already in sorted order. Examples of adaptive sorting algorithms include Natural MergeSort [7], TimSort [26], PowerSort [24], and multiway PowerSort [14]. Among those, the first three algorithms use the two-way merges of runs (i.e., AdaptMerge) from Carlsson et al. [7] as subroutines, whereas the multiway PowerSort employs k-way merges of runs.

Persistent homology is a popular tool from topological data analysis (TDA) that captures the topological features of a space at different scales. In its simplest form, given a real-valued function $f : \mathbb{R} \to \mathbb{R}$, persistent homology computes the pairings among local extrema (i.e., local maxima and local minima) of f. These *persistence pairs* encode the topological features of f at different scales.

In this paper, we combine two seemingly unrelated concepts—adaptive sorting and persistence—and introduce a new sorting algorithm, referred to as the Persistence Sort (PersiSort in short, pronounced "Percy sort"), that utilizes the persistence pairs among local extrema of a list. Our contributions include:

- We provide, for the first time, a general worst-case bound for a class of adaptive sorting algorithms. We prove that any adaptive sorting algorithm that uses AdaptMerge of Carlsson et al. [7] (i.e., two-way merges of runs) performs $\mathcal{O}(nH(\ell_1, \ldots, \ell_r))$ comparisons on a list of n elements containing r precomputed runs each with ℓ_i elements, and is, therefore, worst-case comparison optimal. Here, $H(\ell_1, \ldots, \ell_r) = -\sum_{i=1}^{r} (\ell_i/n) \log(\ell_i/n)$ is the entropy of the runs [2].
- Using PersiSort, we provide a new way to analyze adaptive sorting by looking at the arrangement of runs based on the topological notion of persistence. Unlike previous work such as TimSort and PowerSort, PersiSort does not consider the number of elements but the values of elements in the sorting process.
- We demonstrate that PersiSort outperforms several state-of-the-art adaptive sorting algorithms on data distributions where runs have little overlap in their ranges of values. Our experiments suggest ways to improve PowerSort by using AdaptMerge as its merge subroutine.

Finally, we provide an open-source implementation of PersiSort on Github[1].

2 Related Work

Adaptive sorting algorithms. Any sorting algorithm requires worst-case $\Omega(n \log n)$ comparisons to sort a list of n elements. In particular, MergeSort [18] has a $\Theta(n \log n)$ complexity for all inputs. An adaptive sorting algorithm seeks to use the presortedness of the input to make informed decisions about the merges performed.

The first adaptive sorting algorithm uses the number of inversions $\mathrm{Inv}(X)$ in the input list X as a measure of

*Aarhus University. jkrschou@gmail.com
†University of Utah. beiwang@sci.utah.edu

[1] https://github.com/Nimakii/PersiSort

presortedness. $\mathrm{Inv}(X)$ is the number of pairs of input elements in the wrong order [13]. Given a list X with n elements, Guibas et al. [16] gave the first adaptive sorting algorithm with a complexity $\Omega\left(n \log\left(\frac{\mathrm{Inv}(X)}{n}\right) + n\right)$, using a finger-based balanced search tree. Other adaptive sorting algorithms include BlockSort [21], SplitSort [19], and Adaptive HeapSort [20].

The second type of adaptive sorting algorithm uses *runs* (i.e., presorted subsequences, see Sec. 3.2) as a measure of presortedness. Carlsson et al. [7] introduced AdaptMerge, which uses exponential and binary searches to merge two sorted lists. Given a list X with n elements and r runs, Natural MergeSort [7] detects runs and performs pairwise merges in a balanced way using AdaptMerge, giving a $\Theta(n + n \log r)$ complexity. TimSort [26] puts detected runs on a stack and uses a set of involved rules to decide what and when to merge. TimSort was later shown to have worst-case $\mathcal{O}(n \log n)$ complexity [1] w.r.t. comparison and merge cost. PowerSort [24, 28] is similar to TimSort in the sense that it makes a pass of the input from left to right, and for each new run it detects, it either performs some merges or delays the merges by keeping the runs on a stack. It assigns each adjacent pair of runs a "power" score and applies all delayed merges of higher power. PowerSort has become the standard library sort for CPython since 2022.

Our novel PersiSort algorithm belongs to the second type of adaptive sorting algorithms, where we study the organization of runs in an input list using the notion of persistence [11, 6]. Unlike other adaptive sorting algorithms that focus on the number of elements in each run, PersiSort takes advantage of the values of elements in the sorting process and provides a new perspective on adaptive sorting. Whereas TimSort and PowerSort merge adjacent runs, PersiSort merges runs ordered by persistence pairs.

Persistent homology. Persistent homology is a tool from TDA that captures topological features of data across multiple scales. It has seen a wide range of applications in the study of networks, biological molecules, natural images, time series, etc.; see [9, 23, 25] for introductory texts and surveys. To the best of our knowledge, this is the first time persistence has been utilized in the study of sorting algorithms. The persistent homology of functions from $\mathbb{R}$ to $\mathbb{R}$ is studied in [15] and the windows of [4, 8] are reminiscent but slightly different from the persistence boxes we introduce in Sec. 3.3.

3 Technical Background

3.1 A Review on Persistent Homology

We review the notion of persistent homology in the most straightforward 1-dimensional setting; see [9] for some introductory texts and [10] for a formal treatment.

Let $f : \mathbb{M} \to \mathbb{R}$ be a smooth function defined on a 1-dimensional manifold $\mathbb{M} \subseteq \mathbb{R}$. A point $x \in \mathbb{M}$ is a *critical point* of f if and only if $f'(x) = 0$; otherwise, it is a *regular point*. There are two types of critical points, *local maxima* and *local minima*, which are both *extrema points*. The image of a critical point is a *critical value* of f. A critical point x is *non-degenerate* if $f''(x) \neq 0$. f is a *Morse function* if all its critical points are non-degenerate and have distinct function values. Assume f is a Morse function, the *sublevel set* of f is defined as the pre-image $\mathbb{M}_t := f^{-1}((-\infty, t]) = \{x \in \mathbb{M} \mid f(x) \leq t\}$. To compute the persistent homology, we study the topological changes of $\mathbb{M}_t$ as t increases from $-\infty$ to ∞. This could be considered as the common sweep line idea from computational geometry.

Formally speaking, let m be the number of critical values of f. Let $a_0 < \cdots < a_m$ be a sequence of regular values of f such that each interval (a_i, a_{i+1}) contains exactly one critical value of f. A sublevel set filtration of f is a sequence of sublevel sets connected by inclusions,

$$\mathbb{M}_{a_0} \to \mathbb{M}_{a_1} \to \cdots \to \mathbb{M}_{a_m}.$$

In our setting, 0-dimensional persistent homology studies the topological changes of sublevel sets by applying 0-dimensional homology to this sequence,

$$\mathsf{H}_0(\mathbb{M}_{a_0}) \to \mathsf{H}_0(\mathbb{M}_{a_1}) \to \cdots \to \mathsf{H}_0(\mathbb{M}_{a_n}).$$

The 0-dimensional homology group of a topological space $\mathbb{X}$, denoted as $\mathsf{H}_0(\mathbb{X})$, captures the connected components of $\mathbb{X}$. As t increases, the number of connected components in $\mathbb{M}_t$ only changes when t passes a critical value of f.

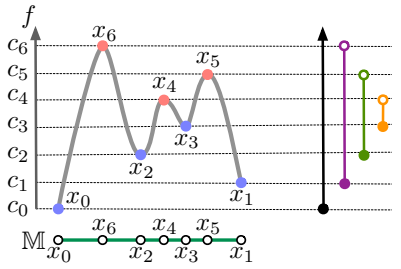


Figure 1: Left: the graph of $f : \mathbb{M} \to \mathbb{R}$, where each point $(x_i, f(x_i))$ is labeled as x_i for simplicity. Right: the 0-dimensional barcode of f based on its sublevel set filtration. Image modified from [29, Fig. 2].

We give an illustrative example in Fig. 1 adapted from [29]. Let x_i denote the critical points and $c_i := f(x_i)$ the critical values of f, ordered as $c_0 < c_1 < \cdots < c_6$ (for readability, we set $c_i = i$). Let $a_0 < a_1 < \cdots < a_7$ be a sequence of regular values of f, where $c_i \in (a_i, a_{i+1})$. As t varies from a_0 to a_7, the 0-dimensional persistent

homology encodes the evolution of connected components in $\mathbb{M}_t$. As illustrated in Fig.1 (left), at $t = a_0$, $\mathbb{M}_t$ is empty. At $t = c_0$, a single component appears in $\mathbb{M}_t$; this is referred to as a *birth* event. At $t = c_1, c_2$, and c_3, a 2nd, 3rd, and 4th component appears in $\mathbb{M}_t$, respectively. At $t = c_4$, the 4th component containing x_3 merges with the 3rd component containing x_2. This is referred to as a *death* event: the younger component containing x_3 disappears (dies) while the elder component containing x_2 remains. Similar death events occur at $t = c_5$ and c_6, respectively. Persistent homology pairs the birth and death events as a set of intervals, called *barcode*, shown in Fig. 1 (right), which contains one infinite bar $[c_0, \infty)$ and three finite bars $[c_1, c_6), [c_2, c_5)$, and $[c_3, c_4)$. Since f is assumed to be a Morse function, critical values of f are unique, and each finite bar in the barcode corresponds to a unique *persistence pair* between a local minimum and a local maximum of f, that is, $[x_1, x_6), [x_2, x_5)$, and $[x_3, x_4)$.

In practice, a smooth function $f : \mathbb{R} \to \mathbb{R}$ may be made into a Morse function using simulation of simplicity (SoS) [12]. It assumes arbitrarily small but not vanishing perturbation to f so that critical points become non-degenerate and have distinct function values (i.e., breaking ties consistently).

3.2 Adaptive Sorting and Run Decomposition

Given a list of elements X, adaptive sorting takes advantage of existing *runs* in the list, which are continuous segments already sorted [1]. However, there are some discrepancies in the definitions of runs. Mannila defined the runs as the ascending segments of X [22], whereas Auger et al. [1] defined a run decomposition as an iterative procedure that builds runs based on the local monotonicity; see Fig. 2 for their differences. Following [1], we could either build a run decomposition from left to right (R_+), or from right to left (R_-), and include local extremum we encounter in the current run. We can also consider assigning local extremum arbitrarily to runs, which we avoid. We work with R_+ in this paper. We further assume that runs are organized in alternating monotonic directions (for persistence, see Sec. 3.1). For example, given $X = [1, 2, 3, 2, 5, 4, 3, 1]$, we have $R_+ = [[1, 2, 3], [2, 5], [4, 3, 1]]$ (from left to right) and $R_- = [[1, 2], [3, 2], [5, 4, 3, 1]]$ (from right to left). R_+ is interpreted to be $R_+ = [[1, 2, 3], [], [2, 5], [4, 3, 1]]$, whereas the 1st and the 3rd runs are monotonically increasing and the 2nd empty run and the 4th run are monotonically decreasing;. However, such an interpretation has no impact on the implementation.

We use the number of comparisons to measure a sorting algorithm's complexity. Specifically, let n be the number of elements in an input list X and r the number of runs. The complexity of a sorting algorithm is the number of element comparisons the algorithm performs as a function of n and r. We have the following known result due to [1, 22].

$$
\begin{aligned}
X &= [12, 10, 7, 5, 7, 10, 14, 25, 36, 3, 5, 11, 14, 15, 21, 22, \\
&\quad 20, 15, 10, 8, 5, 1] \\
R &= [[12, 10, 7, 5], [7, 10, 14, 25, 36], \\
&\quad [3, 5, 11, 14, 15, 21, 22], [20, 15, 10, 8, 5, 1]] \\
R' &= [[12], [10], [7], [5, 7, 10, 14, 25, 36], \\
&\quad [3, 5, 11, 14, 15, 21, 22], [20], [15], [10], [8], [5], [1]]
\end{aligned}
$$

Figure 2: We repeat the example list X from [22] and provide two different run decompositions R and R' of the list X. R takes monotonic continuous segments following [1]. R' consists of increasing continuous segments in line with [22].

Lemma 1 (Adaptive Sorting Lower Bound) *Any adaptive sorting algorithm on an input of size n with r runs has a worst-case comparison complexity of $\Omega(n + n \log r)$ [1, 22].*

The discrepancy described in Fig. 2 can lead to an asymptotic difference in the statement of the lower bound in Lem. 1 as follows. A strictly decreasing sequence of length n would by [22] be decomposed into $r = n$ singleton increasing sequences, while [1] would find the single decreasing sequence for $r = 1$ runs. For this reason, we use the definition of [1].

An adaptive sorting algorithm may be described by a two-step process. First, the algorithm detects all the runs from an input sequence. Second, it merges the runs in some order determined by a merge policy. There are several adaptive sorting algorithms, such as Natural MergeSort [7], TimSort [26], and PowerSort [24]; see App. A for a detailed review on their merge policies.

3.3 Persistence Pairing Among the Extrema of Runs

The persistence pairing among local extrema of a Morse function can be utilized to study relations among the extremal elements of runs in a list, which is at the core of PersiSort. Let X denote a list of n elements, $X = [x_0, \ldots, x_{n-1}]$, where $x_i := X[i]$ (for $0 \le i \le n-1$). For simplicity, assume $x_i \in \mathbb{R}$ and each x_i is unique using the simulation of simplicity (SoS) [12]. X gives rise to a piecewise-linear (PL) function $f : \mathbb{M} \to \mathbb{R}$, where local extrema of f are precisely the extremal elements (extrema) of runs. In other words, the graph of f linearly interpolates among points (i, x_i) (for $0 \le i \le n-1$). Applying sublevel set persistent homology to f gives rise to a persistence pairing among extrema of runs.

Given a list X, a *local maximum* x_i at index i satisfies $x_{i-1} < x_i$ and $x_i > x_{i+1}$. A *local minimum* x_i satisfies $x_{i-1} > x_i$ and $x_i < x_{i+1}$. Depending on their neighbors, x_0 and x_{n-1} are boundary extrema (maximum or

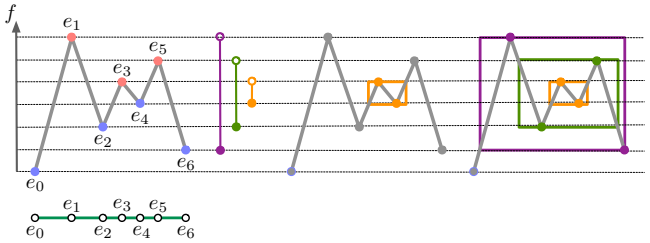


Figure 3: Left: three finite persistence pairs among the extrema: $[e_4, e_3)$, $[e_2, e_5)$, and $[e_6, e_1)$. Middle: a persistence pair $[e_4, e_3)$ with its corresponding persistence box in orange. Right: the nesting of three persistence boxes involving their corresponding persistence pairs.

minimum). Computing persistence pairs of f then gives rise to persistence pairs among the extrema of runs. As shown in Fig. 3 (left), given a list X that contains six extrema (i.e., e_0, e_2, e_4, e_6 are local minima, e_1, e_3, e_5 are local maxima), we obtain three finite persistence pairs: $[e_4, e_3), [e_2, e_5), [e_6, e_1)$; their corresponding bars in the barcode are in orange, green, and purple, respectively.

A persistence pair naturally gives rise to a new concept, a *persistence box*. It is a rectangular box around a persistence pair that is stretched horizontally to include the nearest projections of the local extrema of the pair onto the neighboring runs. See Fig. 3 for an example. Persistence boxes interact with one another, reflecting the relations among runs in a list. A pair of persistence boxes may be *disjoint* or *nested*, or they may *intersect*. In particular, a pair of persistence boxes intersect if their intersection is nonempty and not nested. We describe in Lem. 2 that computing the persistence pairs of X takes linear time.

Lemma 2 *Given a list X of n elements with r runs, the persistence pairs can be computed in $n + \mathcal{O}(r)$ comparisons. If the list of extrema of X is given, then the persistence pairs can be obtained in $\mathcal{O}(r)$ comparisons.*

Proof. Given an element $x_i \in X$, determining whether it is a local extremum relies on its two neighboring elements x_{i-1} and x_{i+1}. Therefore, the local extrema can be computed in a single scan using $n-1$ comparisons.

Assume X contains r runs and E stores the indices of $r+1$ extrema in X. The algorithm to compute persistence pairs proceeds iteratively. During each iteration, it detects pairings among neighboring extrema (referred to as *neighboring persistence pairs*) in E and removes them from the list of extrema E. The algorithm terminates when E is empty or contains one unpaired extremum.

For simplicity, let $e_j := E[j]$ denote an element in the current list of extrema. It has two neighboring extrema e_{j-1} and e_{j+1}. Its *pairing candidate* is one of its neighboring extrema that is closest in terms of its value. Determining the pairing candidate of e_j requires a single comparison between e_{j-1} and e_{j+1}. Two neighboring extrema are paired if they are each other's pairing candidate. The pairing candidate of a boundary extremum is always its neighboring extremum, thus requiring no comparison (e.g., the pairing candidate of e_0 is always e_1 and the pairing candidate of e_r is always e_{r-1}).

Every extremum e_j is removed once from E during some iteration. When e_j is removed, it triggers its neighbor (not paired with e_j) to update its pairing candidate. This takes $\mathcal{O}(1)$ operation. Therefore, processing $r+1$ extrema requires $\mathcal{O}(r)$ comparisons. The comparison complexity is therefore $n + \mathcal{O}(r)$. □

Observation 1 *Intuitively persistence pairings are computed via a sweep line going from $-\infty$ to ∞, but sweep line algorithms require sorting the event points, i.e. extremal values, implying that the comparison complexity of a standard sweep line algorithm would be $\Omega(r \log r)$, excluding the $n-1$ comparisons to find the extremal values. As such, our approach from Lem. 2 is a $\log r$ multiplicative factor improvement.*

We give an illustrative example in Fig. 4. Here, at the beginning of the 1st iteration shown in Fig. 4 (top left), E contains 10 extrema of X, $E = \{e_1, \ldots, e_{10}\}$. The boundary minimum e_0 has a pairing candidate e_1, denoted as $e_0 \to e_1$. The local maximum e_1 has a pairing candidate e_2 since e_2 is closer to e_1 than e_0, therefore $e_1 \to e_2$. Similarly, we have $e_2 \to e_1$, $e_3 \to e_4$, $e_4 \to e_5$, $e_5 \to e_4$, $e_6 \to e_5$, $e_7 \to e_8$, $e_8 \to e_9$, $e_9 \to e_8$, and $e_{10} \to e_9$. Since we have $e_1 \leftrightarrow e_2$, $e_4 \leftrightarrow e_5$, $e_8 \leftrightarrow e_9$ we obtain three neighboring persistence pairs $[e_2, e_1)$, $[e_4, e_5)$, and $[e_8, e_9)$, shown in orange, red, and purple, respectively, see Fig. 4 (top left). Removing the extrema involved in these pairs gives rise to an updated list of extrema at the beginning of the 2nd iteration, $E = \{e_0, e_3, e_6, e_7, e_{10}\}$. The pairing candidates of their neighboring extrema are also updated. See Fig. 4 (top middle).

For instance, as shown in Fig. 4 (top middle), when extrema from the pair $[e_4, e_5)$ are removed from E, we update the pairing candidates of their neighbors e_3 and e_6 to obtain $e_3 \to e_6$ and $e_6 \to e_3$. During the 2nd iteration, we obtain a new neighboring persistence pair $[e_6, e_3)$ since $e_3 \leftrightarrow e_6$. During the 3rd and final iteration, we obtain a final neighboring pair $[e_{10}, e_7)$, shown in teal in Fig. 4 (top right) . Each persistence pair is enclosed by a colored persistence box, shown in Fig. 4 (bottom).

However, corner cases involving the boundary extrema require some care. Based on the algorithm described in Lem. 2, a boundary extremum may be involved in a pair that is not a proper persistence pair. As illustrated in Fig. 5, e_9 is a boundary maximum, but the pair $[e_8, e_9)$ is not a proper persistence pair: e_8 gives rise to a new component, which is not killed at

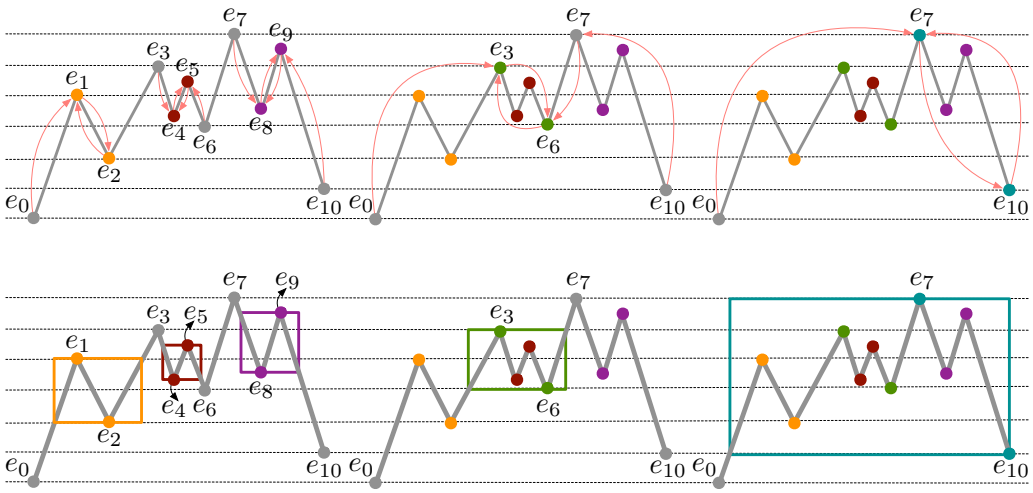


Figure 4: An illustration of computing persistence pairs. Top: a pink arrow from each e_i points to its pairing candidate during each iteration. Bottom: Persistence boxes surrounding persistence pairs were detected during each iteration. From left to right: the 1st iteration returns pairs shown in orange, red, and purple, respectively; the 2nd iteration returns a pair in green; and the 3rd and final iteration returns a pair in teal.

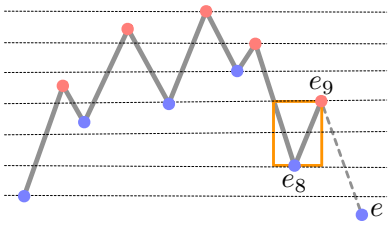


Figure 5: Adding a dummy run (dotted line) for a boundary extremum.

e_9. Such a pair can be made into a proper persistence pair conceptually by adding a dummy run adjacent to the boundary extremum that extends beyond the global minimum. A boundary minimum can be handled similarly by adding an adjacent dummy run that extends beyond the global maximum.

In the case of duplicates, SoS is not actually implemented in PersiSort due to its non-negligible overhead, instead we employ simple rules to handle the pairings in a way that remains consistent with persistence. Specifically, in the case of equally valued pairing candidates, the maximum would first consider the candidate with the smaller index, and the minimum would first consider the candidate with the larger index.

3.4 AdaptMerge and FingerMerge

The key idea behind PersiSort is performing a pair of three-way merges—referred to as FingerMerge—around persistence pairs. Carlsson, Levcopoulos, and Petersson [7] introduced a merging procedure that uses exponential and binary search [3] to achieve the optimal number of comparisons when merging two sorted lists. This is referred to as AdaptMerge in [7], it is also known as galloping, which is employed by TimSort [1, 26]. FingerMerge employs AdaptMerge twice to perform a three-way merging of three sorted lists. We review the idea behind AdaptMerge for completeness. We slightly modify in Fig. 6 an example from [7, Fig. 1]. The input to a merging algorithm consists of two sorted lists, A and B. The output is a sorted list C. Each entry in C is a consecutive subsequence of A or B, e.g., $C[0] = A[0, 4]$ and $C[1] = B[0, 0]$. We obtain the merged sequence by reporting the elements in these subsequences in the order in which they appear in B.

$$\begin{aligned} A =& [1, 2, 3, 4, 5, 7, 8, 11] \\ B =& [6, 9, 10, 12, 13, 14] \\ C =& [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] \\ =& [A[0, 4], B[0, 0], A[5, 6], B[1, 2], A[7, 7], B[3, 5]] \end{aligned}$$

Figure 6: An example of the input and output of a merging algorithm adapted from [7, Fig. 1]. We write $A[i_1, i_2]$ to represent the elements of A at indices i_1 through i_2.

We now describe AdaptMerge applied to two sorted list A and B; w.l.o.g., we assume that $a_0 := A[0] < b_0 := B[0]$. Consider the example in Fig. 6. To compute C, we find the positions in which A and B have to be split, where portions of the other sequence should be inserted. In other words, we compute the elements in A and B that would receive new successors in the resulting sequence. For example, number 5 from A receives a new successor 6 from B in the resulting sequence C; number 10 from B receives a new successor 11 from A in C, and so on. The intuition behind AdaptMerge is that if there are large consecutive portions in A and B in

which all elements would keep their original successor after merging [7], then these elements do not need to be examined entirely during merging. AdapteMerge "uses exponential and binary search to pass such portions as fast as possible in our search for the next element that would receive a new successor" [7].

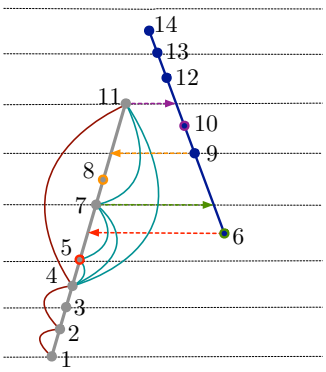


Figure 7: An illustration of AdaptMerge algorithm. The deep red curve illustrates the exponential search (by indices), whereas the teal curve illustrates the binary search. Elements from both sorted lists are laid out in increasing values, resembling a run.

Following [7], let $\{a_{i_0}, a_{i_1}, \ldots, a_{i_p}\}$ be a set of elements in A that will receive new successors. For example, this set equals $\{5, 8, 11\}$ in A. Similarly, $\{b_{j_0}, b_{j_1}, \ldots, b_{j_q}\}$ is a set of elements in B that will receive new successors; this would be $\{6, 10\}$ in B. If A and B are two sorted sequences of length n and m, respectively, we define $\text{Rank}(b_j, A) = max\{\ell \mid 0 \leq \ell \leq n, b_j > a_\ell\}$, for $0 \leq j \leq m-1$. This means the maximum element in A is smaller than (thus closest to) b_j. $\text{Rank}(b_j, A)$ tells us where the split of A and the actual merge from B has to start.

As illustrated in Fig. 7, for the example from Fig. 6, AdaptMerge starts by computing $i_0 = \text{Rank}(b_0, A) = 4$ by an exponential and binary search forward in A based on element indices. Deep red curves illustrate the exponential search, whereas teal curves illustrate the binary search. The red dotted arrow illustrates the computing process i_0, and $A[i_0]$ is highlighted as a red point. Then $C[0] = A[0, i_0] = A[0, 4]$. Second, we compute $j_0 = \text{Rank}(a_{i_0+1}, B) = 0$ by an exponential and binary search forward in B. And we set $C[1] = B[0, j_0] = B[0, 0]$ (see the green dotted arrow and the green point). Third, we compute $i_1 = \text{Rank}(b_{j_0+1}, A) = 6$ by an exponential and binary search forward in X starting from $A[i_0 + 1]$, and set $C[2] = A[i_0 + 1, i_1] = A[5, 6]$ (see the orange dotted arrow and the orange point). We continue to perform exponential and binary searches, alternating between A and B. We start the search from where the last element is found to receive a new successor. When one of the sequences is finished, the next empty entry in C is set to the remaining portion of the nonempty sequence (e.g., $B[3, 5]$ is copied over). For completeness, the pseudocode of AdaptMerge is included in App. B. The following complexity of AdaptMerge is obtained by studying the worst case lower bound on the number of comparisons performed by a merging algorithm in Lem. 3 and Thm. 4 of [7].

Lemma 3 ([7]) *Applying AdaptMerge to two sorted lists of lengths $n_1 \leq n_2$ has a worst-case comparison complexity $\mathcal{O}\left(n_1 \log\left(\frac{n_1+n_2}{n_1}\right)\right)$.*

Observation 2 *In particular, AdaptMerge of two sorted lists where all elements of one list have smaller values than all elements of the other performs $\mathcal{O}(\log n_1)$ comparisons (assuming $n_1 \leq n_2$).*

We define three-way FingerMerge(A,B,C) to be AdaptMerge(AdaptMerge(A,B),C); see App. B for pseudocode for both merge algorithms.

Recall that we study the complexity of a sorting algorithm using the *comparison cost*, also used by the Natural MergeSort and its subroutine AdaptMerge [7]. PowerSort [24] and multiway PowerSort [14], on the other hand, are optimized with regards to the *merge cost*. To merge two runs of lengths n_1 and n_2, AdaptMerge has a merge cost of $\mathcal{O}(n_1 + n_2)$ and a comparison cost of $\mathcal{O}\left(n_1 \log\left(\frac{n_1+n_2}{n_1}\right)\right)$ for $n_1 \leq n_2$. These two costs are equivalent in the worst case. We explore the Merge Tree that encodes the merging order in Sec. 4.

4 New Result: Optimal Bound for Adaptive Sorting

Based on our discussion of AdaptMerge in Sec. 3.4, we prove, for the first time, that any adaptive sorting algorithm that uses AdaptMerge (i.e., two-way merges of runs) as a subroutine is worst-case optimal. In particular, given a list of n elements containing r (precomputed) runs, such an algorithm has a worst-case comparison complexity of $\mathcal{O}(n \log r)$.

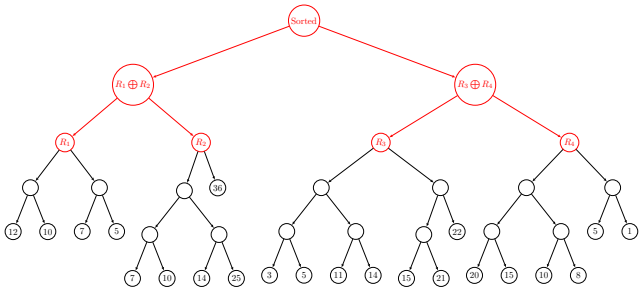


Figure 8: An exemplar Merge Tree of the list in Fig. 2. AdaptMerge(R_1, R_2) is represented by $R_1 \bigoplus R_2$.

To analyze the comparison complexity of a merge-based sorting algorithm, we can view the intermediate

steps as a tree, referred to as a *Merge Tree* of a sorting algorithm[2]. In a Merge Tree, leaves represent (sublists of) single elements, internal nodes represent intermediate sorted sublists, and the root represents the entire sorted list. Given an input list X of n elements, the classic MergeSort first divides X into n (sorted) sublists of one element and then repeatedly merges sublists to produce sorted ones until only one sublist remains. Therefore, the classic MergeSort produces an almost perfectly balanced Merge Tree. On the other hand, any iterative merge-based sorting algorithm that inserts elements into a sorted list one at a time can be represented by a maximally unbalanced Merge Tree.

We extend the notion of a Merge Tree to adaptive sorting by introducing the *Adaptive Merge Tree*, which is a Merge Tree whose leaves represent nonempty sorted lists (runs). In Fig. 8, we represent an Adaptive Merge Tree (in red) as a subtree of a Merge Tree (in red and black) for the example in Fig. 2. Not every Merge Tree contains an Adaptive Merge Tree as a subtree. Here, R_1, R_2, R_3 and R_4 (in red) are the four runs from the run decomposition R in Fig. 2. Each subtree rooted at R_i (in black) is a Merge Tree that shows how (an instance of) a MergeSort would have constructed R_i from the input. Given an input list of n element with r runs, the run decomposition requires $n-1$ comparisons. Using an Adaptive Merge Tree, we produce an upper bound in Thm. 4. Assume that we have an input list of n elements containing r precomputed runs, where an ith run contains ℓ_i elements.

Theorem 4 *Any adaptive sorting algorithm that uses AdaptMerge as a subroutine performs* $\mathcal{O}(nH(\ell_1,\ldots,\ell_r)) = \mathcal{O}(n\log r)$ *comparisons to merge precomputed runs based on its predetermined merge policy. In the case when runs are not precomputed, the comparison complexity is* $n+\mathcal{O}(nH(\ell_1,\ldots,\ell_r)) = n+\mathcal{O}(n\log r)$.

Proof. We assume X to be a list of n elements containing r runs, and each run contains $\ell_i \geq 2$ elements. We assume the runs have been precomputed and the merge policy has been predetermined, therefore we start with an established Adaptive Merge Tree T. Using AdaptMerge as a subroutine, we report on the number of comparisons needed to merge the sublists from the leaves to the root of T.

Let $f : T \to \mathbb{Z}$ be a function that assigns to each node $v \in T$ the number of comparisons performed to reach its corresponding sublist from its children. For any leaf v, set $f(v) = 0$. The comparison complexity of the algorithm is the number of comparisons needed to arrive at the root o of T, that is, $f(o)$. We prove that $f(o) \leq cn(H(\ell_1,\ldots,\ell_r))$ by induction on the size of the tree. Here, the constant $c > 0$ comes from the $\mathcal{O}$ notation of Lem. 3.

If we have a single run ℓ_1, $H(\ell_1) = 0$, which is trivial. We thus start with an base case (BS) with two runs of size ℓ_1 and ℓ_2, where $\ell_1+\ell_2 = n$. This corresponds to an Adaptive Merge Tree T that contains a root o with two leaves that correspond to runs of lengths ℓ_1 and ℓ_2 respectively (w.l.o.g., assuming $\ell_1 \leq \ell_2$). A single AdaptMerge is applied to the two runs and according to Lem. 3, the comparison cost is

$$\begin{aligned} f(o) &\leq c\ell_1 \log\left(\frac{\ell_1+\ell_2}{\ell_1}\right) \\ &< c\ell_1 \log\left(\frac{\ell_1+\ell_2}{\ell_1}\right) + c\ell_2 \log\left(\frac{\ell_1+\ell_2}{\ell_2}\right) \\ &= -c\ell_1 \log\left(\frac{\ell_1}{\ell_1+\ell_2}\right) - c\ell_2 \log\left(\frac{\ell_2}{\ell_1+\ell_2}\right) \\ &= cn\,(H(\ell_1,\ell_2)) \end{aligned}$$

For the induction hypothesis (IH), we assume the bound holds for trees with $r-1$ runs or less.

For the induction step, we need to show the bound holds for trees with r runs. Let T be a given Adaptive Merge Tree and o be its root. Let o_L be the root of the left subtree over n_L elements whose leaves (runs) are indexed by an index set I_L. Similarly, let o_R be the root of the right subtree over n_R list elements whose leaves (runs) are indexed by an index set I_R. By construction, $n_L + n_R = n$, $|I_L| + |I_R| = r$, $\sum_{i\in I_L} \ell_i = n_L$ and $\sum_{j\in I_R} \ell_j = n_R$. Assume *w.l.o.g.* that $n_L \leq n_R$.

The comparison cost needed to reach the root o is obtained from the comparison cost required to reach its children o_L and o_R plus the cost of merging their corresponding sublists, according to Lem. 3. Since $|I_L| \leq r-1$ and $|I_R| \leq r-1$, the IH holds for both the left and right subtree.

To complete the induction we first look at the entropy function $H(h_1,\ldots,h_k)$, denoted as $H(\{h_i\}_{i\in I})$ for simplicity (for all i in the index set I). For $h_i > 0$ and $m = \sum_{i=1}^{k} h_i$, we have,

$$\begin{aligned} mH(h_1,\ldots,h_k) &= \sum_{i=1}^{k} h_i \log\left(\frac{m}{h_i}\right) \\ &= \log(m)\sum_{i=1}^{k} h_i - \sum_{i=1}^{k} h_i \log(h_i) \\ &= m\log(m) - \sum_{i=1}^{k} h_i \log(h_i) \qquad (1) \end{aligned}$$

Returning to our induction and apply Eqn. 1 to the left

[2]This is an entirely different concept from the merge tree of a scalar field commonly used in TDA; see [29] for a survey.

and right subtrees,

$$\begin{aligned}
f(o) &\le f(o_L) + f(o_R) + cn_L \log\left(\frac{n_L + n_R}{n_L}\right) \\
&\le cn_L H(\{\ell_i\}_{i\in I_L}) + cn_R H(\{\ell_j\}_{j\in I_R}) \\
&\quad + cn_L \log\left(\frac{n_L + n_R}{n_L}\right) \\
&= cn_L \log(n_L) - c\sum_{i\in I_L} \ell_i \log(\ell_i) \\
&\quad + cn_R \log(n_R) - c\sum_{j\in I_R} \ell_j \log(\ell_j) \\
&\quad + cn_L \log\left(\frac{n}{n_L}\right) \\
&= cn_R \log(n_R) + cn_L \log(n) - c\sum_{i=1}^{r} \ell_i \log(\ell_i) \\
&< cn_R \log(n) + cn_L \log(n) - c\sum_{i=1}^{r} \ell_i \log(\ell_i) \\
&= cn \log(n) - c\sum_{i=1}^{r} \ell_i \log(\ell_i) \\
&= cnH(\ell_1, \ldots, \ell_r),
\end{aligned}$$

concluding the induction.

Finally, the right-hand side of the upper bound $\mathcal{O}(n \log(r))$ follows from the fact that $H(\ell_1, \ldots, \ell_r)$ is maximized when $\ell_i = n/r$, in which case $nH(n/r, \ldots, n/r) = \sum_{i=1}^{r} n/r \log r = n \log r$. □

5 New Result: Persistence Sort

We now introduce the novel PersiSort algorithm. The key idea is performing a pair of three-way merges—referred to as FingerMerge—around persistence pairs, that is, merging the three consecutive runs that intersect a persistence box (or two successive runs involving boundary extrema). On a high level, the algorithm identifies persistence pairs with multiple iterations (described in the proof of Lem. 2). It applies FingerMerge to runs that intersect each persistence pair.

Given an input list X with n elements in r runs, we first identify the set of extrema E from X. We then compute the initial set of (neighboring) persistence pairs. We repeat the following procedure until the list is sorted, i.e., when there are at most two extrema in E.

1. For each persistence pair:
   - Perform FingerMerge on the two or three runs that intersect the pair.
     - If the pair contains boundary extrema, perform a two-way merge;
     - Otherwise, perform a three-way merge;
   - Remove the pair of extrema from the set of extrema E.
2. Recompute the persistence pairs by updating the pairing candidates (i.e., neighboring extrema that are closest in terms of values).

Using Fig. 9 as an illustrative example (cf., Fig. 4), we first identify the set of extrema $E = \{e_0, \ldots, e_{10}\}$ and denote the ten runs as $R_1, \ldots, R_{10}$. We compute the initial set of neighboring persistence pairs, whose persistence boxes are visualized as colored boxes in (a). During the 1st iteration, we perform a three-way merge (FingerMerge) of runs intersecting the box for each box. For example, we would merge R_1, R_2, R_3 that intersect the orange box defined by the pair $[e_2, e_1)$. We would then merge R_4, R_5, R_6 that intersect the red box defined by the pair $[e_4, e_5)$, followed by merging R_8, R_9, R_{10} that intersect the purple box defined by the pair $[e_8, e_9)$. We would remove the corresponding extrema from E, resulting in $E = \{e_0, e_3, e_6, e_7, e_{10}\}$. We then recompute the persistence pairs among E, producing a single pair $[e_6, e_3)$ whose green persistence box is visualized in (b). During the 2nd iteration, we merge the three runs intersecting the green box (b), remove the corresponding extrema, and update $E = \{e_0, e_7, e_{10}\}$. During the final iteration, we merge the remaining two runs that intersect the teal box (c), producing a sorted list visualized in (d).

To analyze the comparison complexity of PersiSort, we introduce the notion of *dynamic box depth* of an element $x \in X$, which is the number of persistence boxes it belongs to across iterations, denoted as $d(x)$. As shown in Fig. 9 (cf., Fig. 4), an element $x_i \in X$ belongs to the orange box during the 1st iteration, and the teal box in the 3rd iteration, therefore it has a box depth $d(x_i) = 2$. $x_j \in X$ belongs to the orange box in the 1st iteration, the green box in the 2nd iteration, and the teal box in the 3rd iteration, therefore, it has a box depths $d(x_j) = 3$. $x_k \in X$ belongs to the orange box in the 1st iteration, then it dynamically "moves" into the green box during the 2nd iteration (to somewhere close to x_j, not visualized here), and stays within the teal box during the 3rd iteration, therefore $d(x_k) = 3$. Therefore, the dynamic box depth $d(x)$ captures the number of FingerMerge operations an element x will participate in. We then need the following Lem. 5 and Lem. 6.

Lemma 5 *FingerMerge implicitly computes the persistence boxes.*

Proof. We use Fig. 10 to illustrate the relation between FingerMerge and persistence boxes. First, w.l.o.g., assume $[e_{p+1}, e_p)$ is a persistence pair that does not involve a boundary extremum. The pair intersects three runs R_p, R_{p+1}, and R_{p+2} (with number of elements $\ell_p, \ell_{p+1}, \ell_{p+2}$, respectively). Computing the persistence box of such a pair is equivalent to finding the predecessor of e_{p+1} in R_p and the successor of e_p in R_{p+2}.

Using FingerMerge, we apply AdaptMerge to the runs

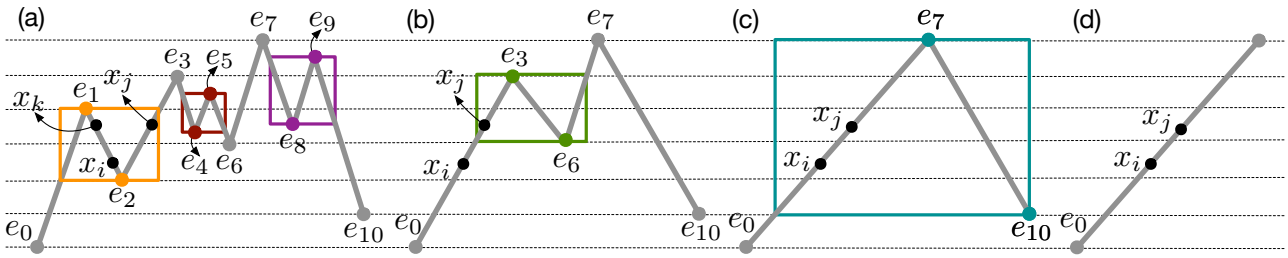


Figure 9: A step-by-step illustration of the PersiSort algorithm.

R_p and R_{p+1}. This process involves finding the maximum element x in R_p that is smaller than e_{p+1} using exponential and binary search (c.f., Fig. 7). x is the location where R_p needs to be split and the actual merge from R_{p+1} has to start. Identifying element x is illustrated by a red dotted arrow from e_{p+1}, and x is shown as a red point. A key observation is that such a process implicitly identifies the lower left corner of the persistence box.

Assume x is located at index i_x in R_p, then the exponential and binary search discovers x in $\mathcal{O}(\log i_x) = \mathcal{O}(\log \ell_p)$ comparisons. After merging R_p with R_{p+1} into R', FingerMerge merges R' with R_{p+2}. In fact, it merges R' starting from index $i_x + 1$ in R' (i.e., the index of e_{p+1} in R'), since all elements in R_{p+2} are larger than e_{p+1}. When R' is exhausted of elements, we have found the successor of e_p in R_{p+2}. This is equivalent to finding the upper right corner of the persistence box, as desired. □

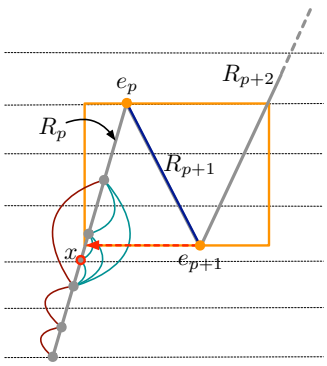


Figure 10: An illustration of the FingerMerge algorithm. The deep red curve illustrates the exponential search, whereas the teal curve illustrates the binary search.

With the above Lemma, we are ready to present an analysis of PersiSort that uses our new computational model. As illustrated in Fig. 9, dynamic box depth is insufficient for analyzing PersiSort since the algorithm requires extra work to detect the boundaries of persistence boxes using FingerMerge. For example, detecting the boundary of the orange box along the run R_p requires $\log(\ell_p)$, which is an overestimation based on the exponential and binary search along R_p. We know from the Adaptive Merge Tree structure that there are r runs represented by leaves, denoted as $R_1, \ldots R_r$; and $r-1$ intermediate sorted sublists represented by internal nodes, denoted as $R_{r+1}, \ldots, R_{2r-1}$. Let $\ell_i = |R_i|$ for $1 \le i \le 2r-1$.

Lemma 6 *Using PersiSort on a list of n elements with r runs, the number of comparisons performed is*

$$n + \mathcal{O}\left(r + \sum_{x \in X} d(x) + \sum_{i=1}^{2r-1} \log \ell_i\right).$$

Proof. First, recall in Lem. 2 that all persistence pairs can be computed in $n + \mathcal{O}(r)$ comparisons, where $n-1$ comparisons are needed to first locate the runs.

Second, the dynamic box depth $d(x)$ of an element $x \in X$ captures the number of FingerMerge operations an element x may participate in. However, x may not be compared during the FingerMerge process. The term $\sum_{x \in X} d(x)$ is thus an overestimation of the contribution of elements to the comparison complexity.

Finally, PersiSort by design performs a FingerMerge on runs intersecting a persistence box during an iteration. We know from Lem. 5 that an element outside a persistence box will contribute logarithmically (in the number of elements outside the box) to the comparison complexity. We can upper bound the number of elements outside of the persistence boxes by the total number of elements across the original and intermediate runs, that is, $\sum_{i=1}^{2r-1} \log \ell_i$. This concludes the comparison complexity analysis of PersiSort. □

Together with Thm. 4, Lem. 6 now provides an alternative comparison complexity analysis of PersiSort.

A desirable property of a sorting algorithms is to be *stable*. Stability is defined as follows: if elements $x = y$, and x precedes y in the initial ordering, then this ordering is preserved after sorting. AdaptMerge is stable, making FingerMerge and subsequently PersiSort stable as well. Although PersiSort does not merge runs sequentially like PowerSort and TimSort, it always merge adjacent runs, which is sufficient for a simple inductive proof.

6 Adaptive Sorting Implementations

We discuss implementation details on several popular sorting algorithms: the sorting algorithm implemented in Python (referred to as Python Sort), TimSort, and PowerSort. We implement PersiSort and TimSort in-house. Following [1, 28], we use the number of comparisons performed to quantify the complexity of these sorting algorithms. We count the number of element comparisons via a custom class for our in-house implementations.

TimSort. We use an in-house Python implementation of TimSort based on the description of [1]. The original TimSort uses galloping, a version of AdaptMerge, discussed in Sec. 3.4.

TimSort is a sequential adaptive sorting algorithm that maintains a stack of runs and applies AdaptMerge to selected pairs of runs. Specifically, it merges the top run and the 2nd run from the top or the 2nd and 3rd runs from the top under a merge policy. It ensures that the sizes of the runs on the stack form an exponentially increasing sequence.

Python Sort. We compare against the sorting algorithm used in Python version 3.11.2[3] at the time of writing. This version of Python implements the PowerSort of Munro and Wild [24].

PowerSort [24] is essentially based on TimSort but with a different merge policy, see App. A. In Python version 3.11.2, PowerSort is the standard library sort, which makes it easy for us to get a comparison. On the other hand, the implementation in Python is highly optimized for time, unlike our in-house PersiSort and TimSort implementations. To differentiate the Python sorting algorithm from the PowerSort described below, this algorithm is referred to as **Python v3.11.2** in our experiments.

PowerSort. Sebastian Wild provided an educational implementation of PowerSort [27] that we used as PowerSort in our experiments. This version is not as highly optimized as the Python standard library version, but we can control which merge subroutine is used. In the original code provided by Wild, a classic $\mathcal{O}(n_1 + n_2)$ merge routine is used (for merging a pair of lists of lengths n_1 and n_2 respectively). For our experiments, we replace it with AdaptMerge to more fairly compare the merge policies of PersiSort and PowerSort.

7 Data Distributions

To empirically compare the adaptive sorting algorithms described in Sec. 6, we introduce six data distributions (also referred to as run configurations) that illustrate different behaviors of the algorithms under scrutiny. The six data distributions include Staircase, Isolated Points, Super Nesting, Uniformly Random, Ultra Nesting, and TimSort Nemesis, presented left to right in Fig. 11. A list sampled from a data distribution is visualized as a PL function: the x-axis represents the indices of list elements, and the y-axis represents their values; elements are visualized as blue points connected by edges following the input order, where runs are easily visible as monotonic segments of the PL curve. We also include one additional data distribution, Overlapping Staircase, that is a variant of Staircase. The lists sampled from these distributions differ by the amount of overlap between runs in their range of values and, subsequently, the dynamic box depths of elements in the lists. We describe the intuition behind these data distributions and the performance of PersiSort on them; see Tab. 12 for an overview.

We sample 100 lists from each data distribution and report the median number of comparisons. For lists sampled from the Staircase, Isolated Points, and Super Nesting distributions, we first vary the number of elements and then the number of runs; see Fig. 13. For lists sampled from Uniformly Random, Ultra Nesting, and TimSort Nemesis, we only vary the number of elements as we cannot control the number of runs; see Fig. 14.

When we vary the number of elements, we hard code 50 runs and let the number of elements range from 150 to 2950 in increments of 100 for a total of 29 data points. Similarly, when we vary the number of runs, we hard code 3000 elements and let the number of runs range from 10 to 750 in steps of 25 for 30 total data points.

Observation 3 (Disjoint Values) *AdaptMerge on two sorted lists A, B of lengths $n_1 \leq n_2$ has a worst-case comparison complexity of $n_1 \log\left(\frac{n_1+n_2}{n_1}\right)$ by Lem. 3. However, if the values in the runs are disjoint, w.l.o.g., assuming $A[n_1 - 1] \leq B[0]$, then AdaptMerge would use worst case $\mathcal{O}(\log n_1)$ comparisons and simply prepends A to B.*

We use Obs. 3 to create data distributions where the benefits of AdaptMerge over regular merge subroutines are maximized.

Staircase. The Staircase distribution is designed such that an element x in the list has a constant dynamic box depth $d(x)$. This is achieved by creating monotonically increasing runs of length three, and monotonically decreasing runs of length $n/(r/2) - 3$ in an alternating fashion, while ensuring that all elements of run i have smaller values than those of run $i + 1$. This ensures that $d(x) = 1$ for all elements. When running PersiSort on the staircase distribution, an accumulator run is created, and the initial runs are merged into it one at a time. This behavior is equivalent to a maximally unbalanced Adaptive Merge Tree. By Obs. 3 and Lem. 6,

[3] `https://www.wild-inter.net/posts/powersort-in-python-3.11`, accessed on December 2, 2023

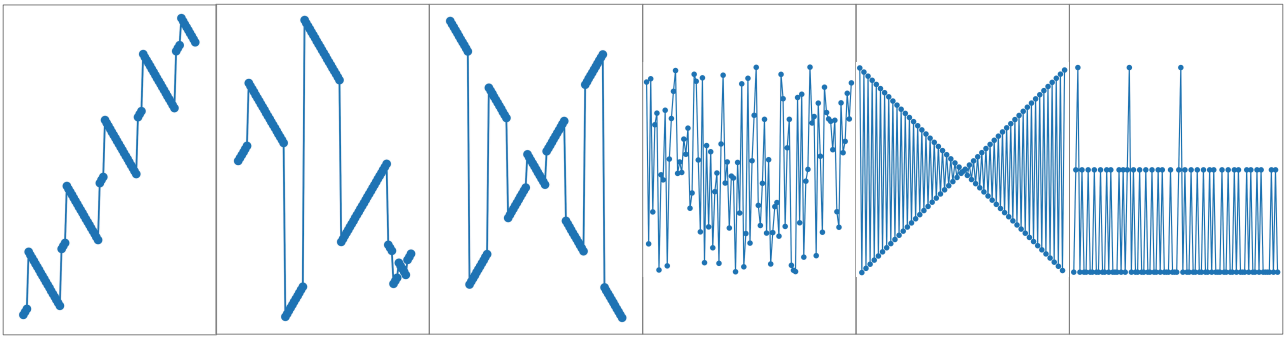


Figure 11: Six types of data distributions. From left to right: Staircase, Isolated Points, Super Nesting, Uniformly Random, Ultra Nesting, and TimSort Nemesis.

Distribution	Disjoint Values	Box depth	PersiSort
Staircase	Yes	1	$\mathcal{O}(n + r\log(rn))$
Isolated Points	Initially	$\mathcal{O}(r)$	$\mathcal{O}(n + n\log r)$
Super Nesting	No	$\mathcal{O}(r)$	$\mathcal{O}(n + r\log n)$
Ultra Nesting ‡	No	$\mathcal{O}(n)$	$\mathcal{O}(n\log n)$
Overlapping Staircase ‡	Variable	$\mathcal{O}(r)$	$\mathcal{O}(n + n\log r)$
Uniformly Random ‡	No	$\mathcal{O}(n)$	$\mathcal{O}(n\log n)$
TimSort Nemesis ‡	No	$\mathcal{O}(n)$	$\mathcal{O}(n\log n)$

Figure 12: Overview of our data distributions and the comparison complexity using PersiSort. n is the number of elements in a list with r runs. "Disjoint Values" means that runs contain values that are disjoint. We did not perform experiments by varying r with distributions marked with a ‡.

the comparison complexity of the merges can roughly be described as

$$\mathcal{O}\left(\sum_{i=1}^{2r-1}\log\left(i\cdot n/r\right)\right) = \mathcal{O}\left(r\log n + r\log r\right)$$

for a total comparison complexity of $\mathcal{O}(n + r\log(rn))$.

Isolated Points. We introduce the Isolated Points distribution to investigate the impact of the disjoint runs on the performance. By partitioning a list of unique integers into r continuous subsets of random sizes and then uniformly shuffling them, we obtain elements in a list of stochastic dynamic box depth. The purpose of this is to show that, in general, the performance of PersiSort is consistent with initially disjoint runs. It follows from basic probability theory that uniformly random data has runs of expected constant length, which means that $r = \Theta(n)$ and PersiSort has a comparison complexity of $\mathcal{O}(n + nH) = \mathcal{O}(n + n\log r)$.

Super Nesting and Ultra Nesting. Super Nesting is designed for most elements to have high dynamic box depth while maintaining non-intersecting runs. A way to envision the distribution is to have an "X" shape and remove elements such that there are r pieces of equal lengths, and the projection onto either axis is injective. The lowest level persistence pair throughout the execution of PersiSort is between the endpoints of the innermost run/piece of the "X" shape. At each level, PersiSort performs $\mathcal{O}(\log n)$ comparisons to append the neighboring pieces for a total comparison complexity of $\mathcal{O}(n + r\log n)$. Ultra Nesting is the most extreme version of Super Nesting, where each run has length two.

Overlapping Staircase. We are interested in exploring how increasing the dynamic box depth $d(x)$ affects the performance of PersiSort. To do so, we generalize the Staircase distribution to the Overlapping Staircase distribution, where adjacent runs have increasing overlap in their ranges of values with a controlled parameter. The overlap parameter s controls the length of the short runs, and by "pulling down" the runs, we have the same s values in three adjacent runs. This means that the benefit of FingerMerge is neutralized. As s tends to n/r, the data become a maximally entangled "zigzag", for instance, $s = n/r = 3$ produces data that look like $1, 2, 3, 2, 1, 0, 1, 2, 3, \ldots$. Here, PersiSort will produce an accumulator and merge adjacent length three lists into it. In this specific case that contains very few unique values, it would be much more effective to count the occurrences of each value using Counting Sort or Bucket Sort.

TimSort Nemesis. It is conjectured by [5, 24] that the worst-case input for TimSort is the following recursive sequence:

$$\mathcal{R}(n) = \begin{cases} \langle n\rangle & \text{if } n \le 3; \\ \mathcal{R}(n/2) :: \mathcal{R}(n/2-1) :: \langle 1\rangle & \text{if } 2|n; \\ \mathcal{R}\left(\frac{n-1}{2}\right) :: \mathcal{R}\left(\frac{n-1}{2}-1\right) :: \langle 2\rangle & \text{otherwise,} \end{cases}$$

where :: denotes list concatenation, $2|n$ means $n \equiv 0 \mod 2$ and $\langle k\rangle$ is the list $[0, 1, 2, \ldots, k-1]$.

PersiSort faces the same difficulties with this distribution as with maximally overlapping staircases, the dynamic box depth of an element in the list is $\mathcal{O}(n)$, which by Lem. 6 gives an $\mathcal{O}(n\log n)$ upper bound.

Uniformly Random. Following the footsteps of [5, 24], we sample real numbers from the interval $[0, 1]$ uniformly randomly. This distribution has very short runs,

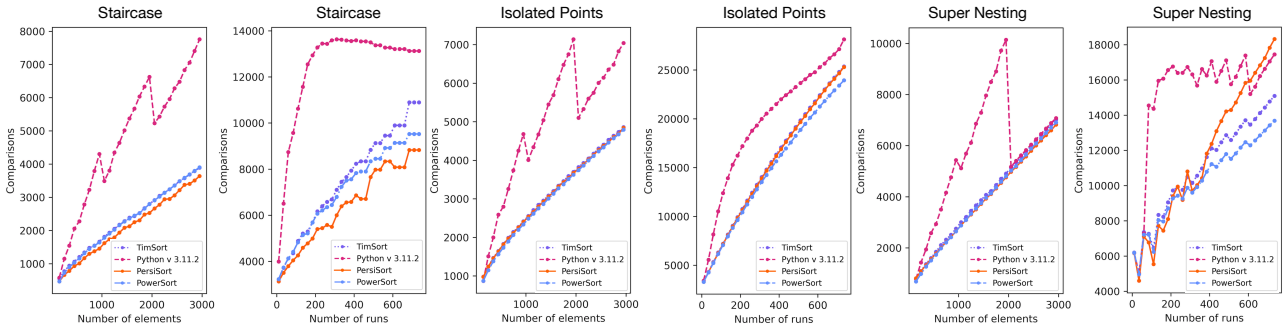


Figure 13: Number of comparisons with Staircase, Isolated Points, and Super Nesting distributions.

in expectation, where the benefit of using an adaptive sorting algorithm is negligible.

8 Experimental Results

We compare the number of comparisons of PersiSort empirically against several adaptive sorting algorithms, including TimSort [1], Python Sort (the PowerSort implementation used in Python version 3.11.2, which does not use AdaptMerge), and PowerSort [24, 27]. Notably, our PowerSort implementation uses AdaptMerge as a subroutine, while Python Sort does not. We use the data distributions described in Sec. 7.

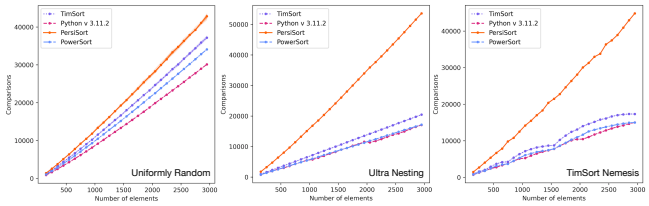


Figure 14: Number of comparisons with Uniformly Random, Ultra Nesting, and TimSort Nemesis distributions.

Highlighted results. As shown in Fig. 13, PersiSort outperforms state-of-the-art adaptive sorting algorithms—PowerSort, TimSort, and Python Sort—on the Staircase data distributions, where runs have no overlap in their ranges of values. This seems reasonable since the merge policy of PersiSort considers the extrema values of the runs. In contrast, TimSort and PowerSort consider the number of elements in the runs when deciding which runs to merge. Meanwhile, PersiSort performs comparably with PowerSort and TimSort on Isolated Points and (partially) on Super Nesting distributions (see Fig. 13). However, it is also clear from Fig. 14 that PersiSort will not replace PowerSort as the standard Python library sorting algorithm. Nevertheless, PersiSort provides a new perspective on adapting sorting based on TDA.

Additional results. We experiment further by increasing the overlap between runs, creating a data distribution from the Overlapping Staircase, see Fig. 15. We create a list of $20,000$ elements in 300 runs and let the overlap between runs vary from 1 through 60 on the x-axis. As the amount of overlap increases, the advantage of PersiSort degrades. We observe an elbow point at $r = 40$ because the dynamic box depth increases drastically from 3 ($r = 40$) to 11 ($r = 60$), increasing the comparison complexity dramatically.

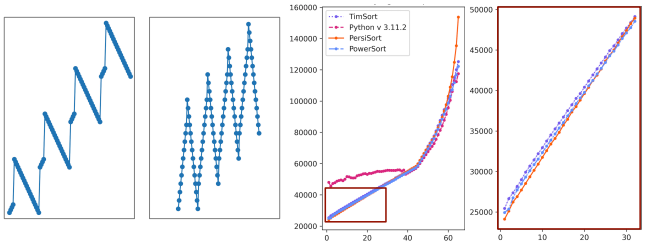


Figure 15: Number of comparisons with the Overlapping Staircase distribution. From left to right: a list sampled from the distribution with a small amount of overlap (before the elbow point); a list sampled from the distribution with a large amount of overlap (after the elbow point); the number of comparisons with increasing number of runs on the x-axis; a zoomed-in view of the red box from the comparison plot.

9 Discussion

PersiSort is well-suited for parallelism, which is left for future work. Introducing parallelism will, however, have no impact on the number of comparisons performed. In cases where the data points have high dynamic box depths, the theoretical performance of AdaptMerge is comparable to that of simpler merge algorithms, which suggests that a highly optimized implementation of AdaptMerge can have practical merit.

We observe in many cases that Python Sort (CPython standard library implementation of PowerSort v 3.11.2) performs equal or worse than our in-house implementation of PowerSort. Our experimental results thus hint

at the possibility of using AdaptMerge as a merge subroutine for Python Sort.

Finally, it is vital for PersiSort that its merge subroutine is AdaptMerge. If a simple linear merge subroutine is used, the comparison complexity becomes $\mathcal{O}\left(n^2\right)$, which is also the worst-case number of element moves performed by PersiSort. It would be interesting to investigate the comparison complexity of PersiSort if AdaptMerge is replaced by other types of merge subroutines (e.g., [17]).

Acknowledgements

JK was supported by the Independent Research Fund Denmark (DFF) grant 9131-00113B. BW was partially supported by grants from National Science Foundation (NSF) DMS-2301361 and IIS-2145499.

References

[1] N. Auger, V. Jugé, C. Nicaud, and C. Pivoteau. On the worst-case complexity of TimSort. In Y. Azar, H. Bast, and G. Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms*, volume 112 of *LIPIcs*, pages 4:1–4:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[2] J. Barbay and G. Navarro. On compressing permutations and adaptive sorting. *Theoretical Computer Science*, 513:109–123, 2013.

[3] J. L. Bentley and A. C.-C. Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976.

[4] R. Biswas, S. Cultrera di Montesano, H. Edelsbrunner, and M. Saghafian. Geometric characterization of the persistence of 1D maps. *Journal of Applied and Computational Topology*, 2023.

[5] S. Buss and A. Knop. Strategies for stable merge sorting. ArXiv preprint arXiv:1801.04641, 2018.

[6] G. Carlsson, A. J. Zomorodian, A. Collins, and L. J. Guibas. Persistence barcodes for shapes. *Proceedings of the Eurographs/ACM SIGGRAPH Symposium on Geometry Processing*, pages 124–135, 2004.

[7] S. Carlsson, C. Levcopoulos, and O. Petersson. Sublinear merging and natural merge sort. In T. Asano, T. Ibaraki, H. Imai, and T. Nishizeki, editors, *Algorithms*, pages 251–260, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[8] S. C. di Montesano, H. Edelsbrunner, M. Henzinger, and L. Ost. Dynamically maintaining the persistent homology of time series. ArXiv preprint arXiv:2311.01115, 2023.

[9] H. Edelsbrunner and J. Harer. Persistent homology - a survey. *Contemporary Mathematics*, 453:257–282, 2008.

[10] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[11] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28:511–533, 2002.

[12] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9:66–104, 1990.

[13] A. Elmasry and M. L. Fredman. Adaptive sorting: an information theoretic perspective. *Acta Informatica*, 45:33–42, 2008.

[14] W. C. Gelling, M. E. Nebel, B. Smith, and S. Wild. Multiway powersort. *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 190–200, 2023.

[15] M. Glisse. Fast persistent homology computation for functions on $\mathbb{R}$. ArXiv preprint arXiv:2301.04745, 2023.

[16] L. J. Guibas, E. M. McCreight, M. F. Plass, and J. R. Roberts. A new representation for linear lists. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 49–60, 1977.

[17] F. K. Hwang and S. Lin. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM Journal on Computing*, 1(1), 1972.

[18] D. Knuth. *The Art Of Computer Programming, vol. 3: Sorting And Searching*. Addison-Wesley, 1973.

[19] C. Levcopoulos and O. Petersson. Splitsort—an adaptive sorting algorithm. *Information Processing Letters*, 39(4):205–211, 1991.

[20] C. Levcopoulos and O. Petersson. Adaptive heapsort. *Journal of Algorithms*, 14(3):395–413, 1993.

[21] C. Levcopoulos and O. Petersson. Exploiting few inversions when sorting: Sequential and parallel algorithms. *Theoretical Computer Science*, 163(1-2):211–238, 1996.

[22] H. Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Transactions on Computers*, C-34(4):318–325, 1985.

[23] E. Munch. A user's guide to topological data analysis. *Journal of Learning Analytics*, 4(2):47–61, 2017.

[24] J. I. Munro and S. Wild. Nearly-optimal mergesorts: Fast, practical sorting methods that optimally adapt to existing runs. In Y. Azar, H. Bast, and G. Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 63:1–63:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[25] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6(17), 2017.

[26] T. Peters. Timsort description. `https://svn.python.org/projects/python/trunk/Objects/listsort.txt`, May 2023.

[27] S. Wild. Educational powersort implementation. `https://colab.research.google.com/drive/13jJDr7dcEz2Ub48TzOT-DaJYCNc5UduR?usp=sharing#scrollTo=KorRRNz_ulvA`.

[28] S. Wild. PyCon US 2023 Talk on Powersort. `https://www.wild-inter.net/posts/powersort-pycon-talk`, 2023.

[29] L. Yan, T. B. Masood, R. Sridharamurthy, F. Rasheed, V. Natarajan, I. Hotz, and B. Wang. Scalar field comparison with topological descriptors: Properties and applications for scientific visualization. *Computer Graphics Forum (CGF)*, 40(3):599–633, 2021.

A Merge Policies of Adaptive Sorting Algorithms

An adaptive sorting algorithm was described by Wild [28] as a two-step procedure. First, the algorithm detects all the runs. Second, it merges the runs in some order determined by a merge policy. The order in which the runs are merged defines a binary tree, referred to as a *Merge Tree*, where leaves represent runs, internal nodes represent intermediate sorted sublists, and the root represents the entire sorted list.

For comparative analysis, we review merge policies of a number of adaptive sorting algorithms. We assume the input is a list with n elements in r runs.

Natural MergeSort [7] has a simple merge policy independent of the runs' values and sizes. Simply put, it builds a (balanced) Merge Tree by merging adjacent runs in the tree. We omit Natural MergeSort from our experiments because it behaves similarly to TimSort.

TimSort [1, 26] determines its merge policy by maintaining a stack of runs. Runs are added to the stack based on the order in which they are discovered. Merging runs from the stack is based on the four rules described below. When the last run is added to the stack, the algorithm collapses the stack by merging runs from top to bottom. The main idea behind these merge triggering rules is that they "balance the run lengths as closely as possible, while keeping a low bound on the number of runs we have to remember" [26].

At any time, let h denote the height of the stack R. Let R_k $(1 \leq k \leq h)$ be the k-th run from the top of the stack, and let $\ell_k := |R_k|$ be the number of elements in R_k. TimSort's merge policy is as follows (based on Algorithm 3 of [1]). Initially, we perform a run decomposition of an input list and set the stack to be empty. At each step of the iteration (until all runs are handled), we remove a run from the run decomposition and push it to the stack. We follow the following four rules to trigger merges:

- If $h \geq 3$ and $\ell_1 > \ell_3$ then merge runs R_2 and R_3;
- else if $h \geq 2$ and $\ell_1 > \ell_2$ then merge runs R_1 and R_2;
- else if $h \geq 3$ and $\ell_1 + \ell_2 \geq \ell_3$ then merge runs R_1 and R_2.
- else if $h \geq 4$ and $\ell_2 + \ell_3 \geq \ell_4$ then merge runs R_1 and R_2.

The analysis of TimSort was proved to be difficult. Auger et al. [1] showed that the runs on the stack are of exponentially increasing size, and TimSort performs $\mathcal{O}(n + n \log r)$ comparisons.

PowerSort was introduced by Munro and Wild [24] and is currently used in Python version 3.11.2. It follows the sequential left-to-right nature of TimSort with some changes in the merge policy. Let R_1 and R_2 be two adjacent runs of lengths ℓ_1 and ℓ_2 respectively. They start at list indices i_1 and i_2 respectively, that is, $R_1 = X[i_1, i_1 + \ell_1 - 1]$ and $R_2 = X[i_2, i_2 + \ell_2 - 1]$. The *power* of the boundary between R_1 and R_2 is defined as

$$p(R_1, R_2) = \max\{\ell \in \mathbb{N} : \lfloor 2^\ell \cdot (i_1 + \ell_1/2)/n \rfloor = \lfloor 2^\ell \cdot (i_2 + \ell_2/2)/n \rfloor\}$$

Similarly to TimSort, PowerSort scans the runs. Assuming there is a run stack $R_0, R_1, \ldots$, and we have discovered a new run R. The algorithm compares the power between R and runs in the stack. If $p(R_0, R) < p(R_0, R_1)$, then R_0 is popped from the stack and merged with R, resulting in R'. Moving forward, if $p(R_0, R) < p(R_1, R_2)$ then R_1 is popped and merged into R'' and so on. Like TimSort, when there are no more runs to process, the stack is collapsed into a single sorted list by merging runs from top to bottom. The comparison complexity of PowerSort [24] is $\mathcal{O}(n + n \log r)$.

B Pseudocode

We provide pseudocode for AdaptMerge, FingerMerge, and PersiSort. The pseudocode of AdaptMerge is included in Fig. 16. By convention, $A[3, 2]$ is an empty list, and $A[3, 3]$ is a single element list. Under these conventions, exponential search for, say, 5 in the list $B = [7, 8, 9]$ returns index -1 such that no elements of B are added to the output C. In the next iteration, the algorithm searches for the predecessor of 7 in, say, $A = [5, 6, 10]$, which will find the predecessor 6 at index 1 and extend C by $A[0, 1] = [5, 6]$. Furthermore, if the exponential part of the exponential search overshoots the end of a list, then it should "round down" to the end of the list and continue to the binary search phase.

```
1  AdaptMerge(A, B):
2      C = empty list of length n_a + n_b
3      i_0 = j_0 = 0
4      while not (i_0 == n_a - 1 or j_0 == n_b - 1):
5          i_1 = ExponentialSearch(B[j_0], A[i_0, n_a - 1])
6          C.extend(A[i_0, i_1])
7          i_0 = i_1
8          j_1 = ExponentialSearch(A[i_0], B[j_0, n_b - 1])
9          C.extend(B[j_0, j_1])
10         j_0 = j_1
11     C.extend(A[i_0, n_a])
12     C.extend(B[j_0, n_b])
13     return C
```

Figure 16: Pseudocode for the AdaptMerge algorithm of Carlsson et al. [7].

The three-way FingerMerge calls AdaptMerge twice, as shown in Fig. 17. The algorithm also needs to ensure that the monotonicity of A, B, C, and AB is the same, which can easily be solved with start and end pointers to the lists. This, however, makes FingerMerge ill-suited for algorithms where reverses are costly.

```
1  FingerMerge(A, B, C):
2      AB = AdaptMerge(A, B)
3      return AdaptMerge(AB, C)
```

Figure 17: Pseudocode for the FingerMerge algorithm.

The pseudocode of PersiSort is shown in Fig. 18. Given an input list X with n elements in r runs, we denote the

sequence of runs as $R_0, R_1, \ldots, R_{r-1}$. We first identify the set of extrema E from X (line 2). We then compute the initial set of (neighboring) persistence pairs (line 3). We repeat the following procedure until the list is sorted, i.e., when there are at most two extrema in E (line 4).

1. For each persistence pair (line 5):
   - Perform FingerMerge on the two or three runs intersecting the pair (lines 6-11).
     - If the pair contains boundary extrema, perform a two-way merge (lines 6-9);
     - Otherwise, perform a 3-way merge (line 11);
   - Remove the pair of extrema from the set of extrema E (line 12).
2. Recompute the persistence pairs by updating the pairing candidates (line 13).

```
1  PersistenceSort(X)
2      E = DetectExtrema(X)
3      PersistencePairs = ComputePairs(E)
4      while |E| > 2:
5          for pair in PersistencePairs:
6              if e_0 in pair:
7                  AdaptMerge(R_0, R_1)
8              elif e_{r-1} in pair:
9                  AdaptMerge(R_{-2}, R_{-1})
10             else:
11                 FingerMerge(R_{pair-1}, R_{pair}, R_{pair+1})
12         E.remove(pair)
13         PersistencePairs = RecomputePairs(E, X)
```

Figure 18: The pseudocode for the PersiSort algorithm. R_{-1} means the last run, and R_{-2} is the second to last run. R_{pair} is the run in the current configuration that contains the persistence pair, and $R_{pair\pm1}$ indicate the runs before and after the current run, respectively.

Flips in Odd Matchings*

Oswin Aichholzer† Anna Brötzner‡ Daniel Perz§ Patrick Schnider¶

Abstract

Let $\mathcal{P}$ be a set of $n = 2m+1$ points in the plane in general position. We define the graph $GM_{\mathcal{P}}$ whose vertex set is the set of all plane matchings on $\mathcal{P}$ with exactly m edges. Two vertices in $GM_{\mathcal{P}}$ are connected if the two corresponding matchings have $m-1$ edges in common. In this work we show that $GM_{\mathcal{P}}$ is connected.

1 Introduction

Reconfiguration is the process of changing a structure into another—either through continuous motion or through discrete changes. Concentrating on plane graphs and discrete reconfiguration steps of bounded complexity, like exchanging one edge of the graph for another edge such that the new graph is in the same graph class, a single reconfiguration step is often called an *edge flip*. The *flip graph* is then defined as the graph having a vertex for each configuration and an edge for each flip. Flip graphs have several applications, for example morphing [6] and enumeration [8]. Three questions are central: studying the connectivity of the flip graph, its diameter, and the complexity of finding the shortest flip sequence between two given configurations. The topic of flip graphs has been well studied for different graph classes like triangulations [3, 15, 16, 17, 18, 20, 21], plane spanning trees [11, 12], plane spanning paths [2, 5], and many more. For a nice survey see [10].

For matchings usually other types of flips were considered since a perfect matching cannot be transformed to another perfect matching with a single edge flip. A natural flip in perfect matchings is to replace two matching edges with two other edges, such that the new graph is again a perfect matching. These flips were studied mostly for convex point sets [9, 19]. While the according flip graph is connected on convex point sets it is open whether this flip graph is connected for any set of points in general position. Other types of flips in perfect matchings can be found in [1, 4, 7].

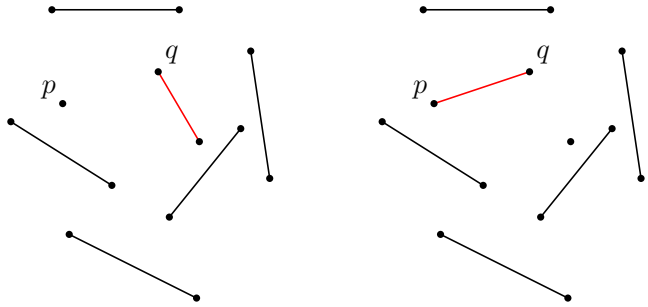


Figure 1: Flipping a matching edge: the previously unmatched point p is matched to q.

In this work we study a setting where single edge flips are possible for matchings. Let $\mathcal{P}$ be a set of $n = 2m+1$ points in the plane in general position (that is, no 3 points are collinear). An *almost perfect matching* on $\mathcal{P}$ is a set M of m line segments whose endpoints are pairwise disjoint and in $\mathcal{P}$. The matching M is called *plane* if no two segments cross.

Let $\mathcal{M}_{\mathcal{P}}$ denote the set of all plane almost perfect matchings on $\mathcal{P}$. We define the flip graph $GM_{\mathcal{P}}$ with vertex set $\mathcal{M}_{\mathcal{P}}$ through the following flip operation. Consider a matching M_1 and let p be the unmatched point. Let $q \neq p$ be a point in $\mathcal{P}$ such that the segment pq does not cross any segment in M_1. The flip now consists of removing the segment incident to q from the matching and adding pq instead, see Figure 1. Note that this gives another plane almost perfect matching M_2. In the graph $GM_{\mathcal{P}}$, the vertices corresponding to M_1 and M_2 are adjacent.

In this paper, we prove the following theorem.

Theorem 1 *For any set $\mathcal{P}$ of $n = 2m+1$ points in general position in the plane the flip graph $GM_{\mathcal{P}}$ is connected.*

In Section 2 we give an overview of the used techniques and the proof of Theorem 1. Then, in Section 3 we prove the lemmata used for the proof of Theorem 1.

2 Overview and Proof of Theorem 1

In this section, we give an overview of our used techniques and the proof of Theorem 1.

*Research on this work has been initiated at the 18th European Geometric Graph Week which was held from September 4^{th} to 8^{th} 2023 in Alcalá de Henares. We thank the organizers and all participants for the good atmosphere as well as for inspiring discussions on the topic.

†Institute for Software Technology, Graz University of Technology, oaich@ist.tugraz.at

‡Department of Computer Science and Media Technology, Malmö University, anna.brotzner@mau.se

§danielperz@gmx.at

¶Department of Computer Science, ETH Zürich, patrick.schnider@inf.ethz.ch

Let $G = (V, E)$ be a graph G and let M be a matching in G. We call a path P in G an *alternating path* if the edges of P lie alternately in M and in $E \setminus M$. A plane path that alternately consists of matching and non-matching edges and connects to the unmatched point gives rise to a sequence of flips, see Figure 2 for an example.

To find such a path, we consider so-called *segment endpoint visibility graphs*: graphs that encode the visibility between the endpoints of a set of segments. More precisely, given a set S of (non-intersecting) segments in the plane, its segment endpoint visibility graph is the graph that contains a vertex for every segment endpoint, and an edge between two vertices if the corresponding segment endpoints either (1) are connected by a segment in S, or (2) "see" each other, meaning that the open segment between them does not intersect any segment from S. Hoffmann and Tóth [13] proved that segment endpoint visibility graphs always admit a simple Hamiltonian polygon—this is a plane Hamiltonian cycle—, and moreover presented an algorithm to find such a polygon. This result is crucial for us, as a plane perfect matching can be considered as a set of segments in the plane. Hence, for every plane matching M there exists a plane subgraph of the segment endpoint visibility graph of M that is the (not necessarily disjoint) union of a Hamiltonian cycle and M.

Even disregarding planarity, we show

Lemma 2 *Let G be an undirected graph that is the union of a Hamiltonian cycle C and a perfect matching M. Let $e = ab$ be a matching edge and let c be any vertex different from a. Then there exists an alternating path P that starts with the vertex a and the edge e and ends with the vertex c.*

The proof of this lemma, and the one of Lemma 3 stated below, are postponed to Section 3.

We denote the *symmetric difference* of two graphs A, B with $A \triangle B$. Given the setup of Lemma 2, we can compute another matching $M_2 = M \triangle P$, in which a is unmatched. This modification corresponds to a sequence of flips in a point set of odd size, see Figure 2. This flip sequence starts with the matching $M = M_1$ and point c being unmatched, and ends with the matching M_2 and point a being unmatched.

To prove that the flip graph $GM_{\mathcal{P}}$ is connected, we show that there always exists a sequence of flips that transforms a given plane almost perfect matching into another plane almost perfect matching, where we may choose any fixed point to be the unmatched point.

Lemma 3 *Let M_1 be a plane almost perfect matching and let t be an arbitrary point of $\mathcal{P}$. Then there exists a sequence of flips to a matching M_2 in which the unmatched point is t.*

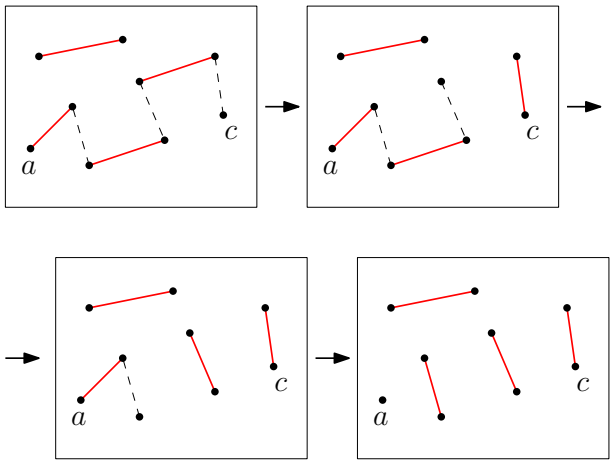


Figure 2: A plane alternating path in the visibility graph gives rise to a sequence of flips.

We use Lemma 3 to show that we can flip every matching M to a *canonical matching* M_C, which we define in the following way. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_{2m+1}\}$, where the points are labeled from left to right. The canonical matching M_C consists of the edges $p_1p_2, p_3p_4, \ldots, p_{2m-1}p_{2m}$ with p_{2m+1} remaining unmatched. It follows from the ordering of the points that this matching is plane.

Proof. [Proof of Theorem 1] Let M be any plane almost perfect matching on $\mathcal{P}$. Let $i \geq 1$ be the smallest index for which the edge $p_{2i-1}p_{2i}$ is not in M. We show that there is a sequence of flips on the point set $\{p_{2i-1}, p_{2i}, \ldots, p_{2m}, p_{2m+1}\}$ after which $p_{2i-1}p_{2i}$ is in the resulting matching. In the following, for simplicity of notation, we set $i = 1$.

Using Lemma 3, we first flip to a matching M_2 in which the point p_1 is unmatched. As the segment p_1p_2 is not crossed by any other segment, we perform one more flip which puts p_1p_2 into the resulting matching. Now we inductively continue the argument on the point set $\mathcal{P}' = \{p_3, \ldots, p_{2m+1}\}$ and eventually reach the canonical matching M_C. Since thus any matching can be transformed to M_C and because the direction of a sequence of flips can be reverted, the statement of Theorem 1 follows. □

3 Proofs of the Lemmata

In this section, we prove Lemma 2 and Lemma 3. We begin this section with presenting a procedure to find an alternating path in an abstract graph. Note that we do not require the path to be Hamiltonian.

Lemma 2 also follows from Lemma 6 in [14]. For the sake of exposition, we give an alternative proof of the same fact, which is arguably simpler.

Lemma 2 *Let G be an undirected graph that is the union of a Hamiltonian cycle C and a perfect matching M. Let $e = ab$ be a matching edge and let c be any vertex different from a. Then there exists an alternating path P that starts with the vertex a and the edge e and ends with the vertex c.*

Proof. In a first step, we reduce to the situation where no matching edge except possibly e lies on the cycle C, that is, $C \cap M \subseteq \{e\}$. To this end, assume that there is a matching edge u_1u_2 lying on the path $\{u_0, u_1, u_2, u_3\}$ of the cycle C. We define the graph G' with vertex set $V(G') = V(G) \setminus \{u_1, u_2\}$ by keeping all edges of G induced by $V(G')$ and adding the edge u_0u_3. It follows from the construction that G' is again the union of a Hamiltonian cycle and a perfect matching and that G' contains an alternating path starting at a and ending at c if and only if G contains an alternating path starting at a and ending at c. As we can iterate this process, in the following we may assume that $C \cap M \subseteq \{e\}$.

We now describe an algorithm that explicitly constructs a required alternating path. The algorithm constructs a sequence of graphs $G_2, G_3, \ldots, G_K$, starting with $G_2 = \{e\}$, with the following properties:

(1) the graph G_k has k vertices $v_1, \ldots, v_k$;

(2) G_k has two vertices of degree 1, namely v_1 and v_k;

(3) all other vertices of G_k have degree 2 and are incident to one edge in M and one edge in $C \setminus M$;

(4) $v_1 = a$, $v_2 = b$ and $v_K = c$.

From these properties it follows that the last graph G_K is the disjoint union of cycles and the required alternating path P. Let us again point out that we do not require the alternating path to be Hamiltonian. It remains to describe the algorithm and prove that the constructed sequence of graphs satisfies the above properties. We start by setting $G_2 = \{e\}$, which trivially satisfies all the properties. In order to construct G_{k+1} from G_k we distinguish two cases, depending on whether in G_k the (unique) edge $\tilde{e}$ incident to v_k is in M or not.

Case 1: $\tilde{e} \in C \setminus M$. Let $m = (v_k, w)$ be the matching edge incident to v_k. We define G_{k+1} by adding m to G_k. By Property (3) for G_k, all vertices in G_k except v_k are incident to an edge in M, and as M is a perfect matching, this implies that w is not a vertex of G_k. Thus, G_{k+1} has one more vertex, proving Property (1) for G_{k+1}. The only vertices whose degrees have changed are $w = v_{k+1}$, which now has degree 1, and v_k which is now also incident to an edge in M. This proves Properties (2) and (3).

Case 2: $\tilde{e} \in M$. For an illustration of this case, see Figure 3. Consider the unique path Q in C from v_k to c which does not pass through a and let w be the first vertex on this path that is not a vertex of G_k.

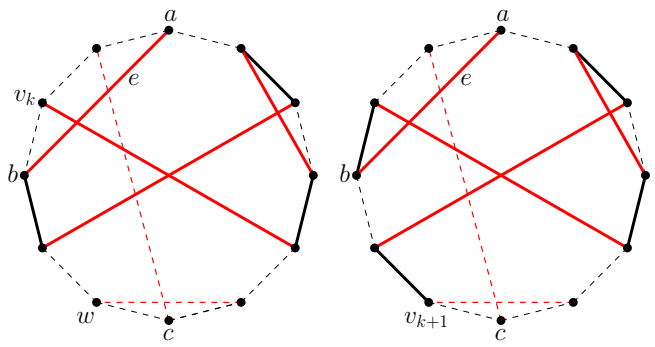


Figure 3: Constructing G_{k+1} (right) from G_k (left). The paths G_k and G_{k+1} are depicted with lines, while unused edges of G are dashed. The matching edges are red, the cycle edges are black.

Let furthermore Q' be the subpath of Q starting at v_k and ending at w. Set $v_{k+1} = w$. For any edge e' in Q', add e' to G_{k+1} if and only if it is not in G_k and remove it otherwise. Properties (1) and (2) follow directly by definition. For Property (3), note that the only vertices whose neighborhoods have changed are the vertices on Q'. As Q' is a path on C and we assumed that C contains no matching edge other than e, it follows that no matching edge was removed. All vertices are thus still incident to exactly one matching edge. Further, as C is a cycle, every vertex in Q' is incident to exactly two edges in $C \setminus M$. It follows from the construction that exactly one of these edges is removed while the other one is added, proving Property (3).

Finally, we stop the procedure as soon as we add the vertex c, which has to happen for some G_K, $K \leq n$, where n is the number of vertices of G. This proves the last part of Property (4) and thus finishes the proof. □

We can now make use of such an alternating path to prove Lemma 3.

Lemma 3 *Let M_1 be a plane almost perfect matching and let t be an arbitrary point of $\mathcal{P}$. Then there exists a sequence of flips to a matching M_2 in which the unmatched point is t.*

Proof. Let p be the unmatched point in M_1. If $p = t$ then we are trivially done, so assume for the remainder that $p \neq t$. We duplicate p such that the two points p, p' have the same neighborhood in the segment endpoint visibility graph. Moreover, we add the edge pp' to M_1. By Theorem 1 in [13], there is a plane Hamiltonian cycle C that spans all segment endpoints of M_1. Moreover, $M_1 \cup C$ is plane. Let u be the vertex that is matched to t in M_1. By Lemma 2, there is an alternating path P from t to p in $C \cup M_1$ that starts with the edge tu. Since the underlying graph is plane, P is also plane. If p and p' are in P, then the edge pp' is also in P because pp' is

a matching edge. Hence, we can contract p and p' to a single point p such that P is still an alternating path.

Now, we construct a matching M_2 by transforming M_1 via a sequence of flips along P to get $M_2 = M_1 \triangle P$. M_2 is an almost perfect matching in which p is matched, and t is the unmatched point. □

The crux in the proof of Lemma 3 lies in safely duplicating the unmatched point p. Both points shall see the same segment endpoints, while visibility between the segment endpoints must not be blocked. This can be achieved by constructing the cell arrangement induced by the lines through all pairs of segment endpoints, and then placing the duplicated point p' in the cell in which p is located. Since the new segment pp' does not intersect any line of the line arrangement, it does not intersect any edge of the segment endpoint visibility graph either. Moreover, any segment endpoint that sees p sees the entire cell in which p is located, and in particular it also sees p'.

4 Remarks on the Diameter of the Flip Graph

From the proof of Theorem 1 it follows directly that no more than $O(n^2)$ flips are needed to transform any plane almost perfect matching on $\mathcal{P}$ into any other plane almost perfect matching on $\mathcal{P}$. In other words, the diameter of the flip graph $GM_{\mathcal{P}}$ is in $O(n^2)$.

On the other hand, flipping from one plane almost perfect matching M_1 to another plane almost perfect matching M_2 may take linearly many steps. To see this, consider two disjoint matchings M_1 and M_2. Each flip adds at most one edge of the target matching M_2, so it takes at least m flips to transform M_1 into M_2.

Moreover, it may take linearly many flips to reach a matching where the unmatched point lies on the boundary of the convex hull, as can be seen in the example in Figure 4. We need to flip edges on each layer of the given matching M_1. However, observe that this is only one sequence of flips during the procedure of transforming M_1 to the canonical matching M_C, and after removing the first edge, the unmatched point will already be closer to the boundary of the convex hull. This indicates that there might exist a procedure that takes fewer than $O(n^2)$ flips to transform any given matching to the canonical matching M_C.

5 Conclusion

We considered the flip graph $GM_{\mathcal{P}}$ of plane matchings for point sets of odd size, and showed that $GM_{\mathcal{P}}$ is connected. In the course of the proof, we also showed that the union of a Hamiltonian cycle and a perfect matching always contains an alternating path from an arbitrary matching edge to any arbitrary point.

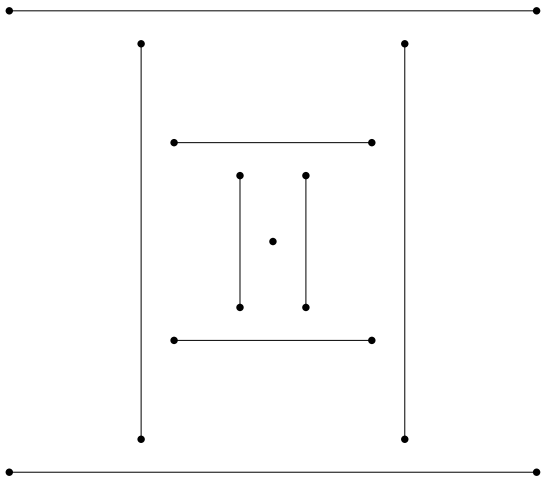


Figure 4: It takes $\Omega(n)$ flips to transform the given matching to any matching where the unmatched point is on the boundary of the convex hull.

While we showed that the flip graph is connected, it would be interesting to determine more precise bounds for the diameter of $GM_{\mathcal{P}}$.

Finally, the question of connectedness remains open for the flip graph of plane perfect matchings on point sets of even size, where in each flip exactly two edges are replaced.

References

[1] O. Aichholzer, S. Bereg, A. Dumitrescu, A. García, C. Huemer, F. Hurtado, M. Kano, A. Márquez, D. Rappaport, S. Smorodinsky, D. Souvaine, J. Urrutia, and D. Wood. Compatible geometric matchings. *Computational Geometry*, 42(6-7):617–626, 2009.

[2] O. Aichholzer, K. Knorr, W. Mulzer, J. Obenaus, R. Paul, and B. Vogtenhuber. Flipping plane spanning paths. In *International Conference and Workshops on Algorithms and Computation*, pages 49–60. Springer, 2023.

[3] O. Aichholzer, W. Mulzer, and A. Pilz. Flip Distance Between Triangulations of a Simple Polygon is NP-Complete. *Discrete Comput. Geom.*, 54(2):368–389, 2015.

[4] O. Aichholzer, J. Obmann, P. Paták, D. Perz, J. Tkadlec, and B. Vogtenhuber. Disjoint compatibility via graph classes. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 16–28. Springer, 2022.

[5] S. G. Akl, M. K. Islam, and H. Meijer. On planar path transformation. *Information Processing Letters*, 104(2):59–64, 2007.

[6] S. Alamdari, P. Angelini, F. Barrera-Cruz, T. M. Chan, G. D. Lozzo, G. D. Battista, F. Frati, P. Haxell, A. Lubiw, M. Patrignani, V. Roselli, S. Singla, and B. T.

Wilkinson. How to morph planar graph drawings. *SIAM Journal on Computing*, 46(2):824–852, 2017.

[7] G. Aloupis, L. Barba, S. Langerman, and D. L. Souvaine. Bichromatic compatible matchings. *Computational Geometry*, 48(8):622–633, 2015.

[8] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.

[9] A. Biniaz, A. Maheshwari, and M. Smid. Flip distance to some plane configurations. *Computational Geometry*, 81:12–21, 2019.

[10] P. Bose and F. Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009.

[11] N. Bousquet, L. De Meyer, T. Pierron, and A. Wesolek. Reconfiguration of plane trees in convex geometric graphs. *arXiv preprint arXiv:2310.18518*, 2023.

[12] M. C. Hernando, F. Hurtado, A. Márquez, M. Mora, and M. Noy. Geometric tree graphs of points in convex position. *Discrete Applied Mathematics*, 93(1):51–66, 1999.

[13] M. Hoffmann and C. D. Tóth. Segment endpoint visibility graphs are Hamiltonian. *Computational Geometry*, 26(1):47–68, 2003.

[14] M. Hoffmann and C. D. Tóth. Alternating paths through disjoint line segments. *Information Processing Letters*, 87(6):287–294, 2003.

[15] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, pages 333–346, 1999.

[16] I. Kanj, E. Sedgwick, and G. Xia. Computing the flip distance between triangulations. *Discrete & Computational Geometry*, 58(2):313–344, 2017.

[17] C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

[18] A. Lubiw and V. Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry*, 49:17–23, 2015.

[19] M. Milich, T. Mütze, and M. Pergel. On flips in planar matchings. *Discrete Applied Mathematics*, 289:427–445, 2021.

[20] A. Pilz. Flip distance between triangulations of a planar point set is APX-hard. *Computational Geometry*, 47(5):589–604, 2014.

[21] U. Wagner and E. Welzl. Connectivity of triangulation flip graphs in the plane. *Discrete & Computational Geometry*, 68(4):1227–1284, 2022.

Finding maximum matchings in RDV graphs efficiently

Therese Biedl* Prashant Gokhale †

Abstract

In this paper, we study the maximum matching problem in *RDV graphs*, i.e., graphs that are **v**ertex-intersection graphs of **d**ownward paths in a **r**ooted tree. We show that this problem can be reduced to a problem of testing (repeatedly) whether a vertical segment intersects one of a dynamically changing set of horizontal segments, which in turn reduces to an orthogonal ray shooting query. Using a suitable data structure, we can therefore find a maximum matching in $O(n \log n)$ time (presuming a linear-sized representation of the graph is given), i.e., without even looking at all edges.

1 Introduction

The MATCHING problem is one of the oldest problems in the history of graph theory and graph algorithms: Given a graph $G = (V, E)$, find a *matching* (a set of pairwise non-adjacent edges) that is *maximum* (has the largest possible number of edges). See for example extensive reviews of the older history of matchings and its applications in [2, 22]. The fastest known algorithm for general graphs runs in $O(\sqrt{n}m)$ time ([25], see also [32]). There have been some recent break-throughs for algorithms for maximum flow, culminating in an algorithm with almost-linear run-time $O(m^{1+o(1)})$ [6]; this immediately implies an almost-linear algorithm for MATCHING in bipartite graphs. See also [7] for a purely combinatorial almost-linear algorithm for the same problem.

Greedy-algorithm and interval graphs. Naturally one wonders whether truly linear-time algorithms (i.e., with $O(m + n)$ run-time) exist, at least if the graphs have special properties. One natural approach for this is to use the greedy-algorithm for MATCHING shown in Algorithm 1, which clearly takes linear time. With a suitable vertex order this will always find the maximum matching (enumerate the vertices so that matched ones appear consecutively at the beginning); the challenge is hence to find a vertex order (without knowing the maximum matching) for which the greedy-algorithm is guaranteed to work.

Algorithm 1: Greedy-algorithm for matching

```
Input: A graph G with a vertex order v_1, ..., v_n
Initialize the matching M = ∅
for i = 1, ..., n do
    if v_i is not yet matched and has unmatched neighbours then
        among all unmatched neighbours of v_i, let v_j be the one that minimizes j
        // j > i, for otherwise v_j would have been matched earlier
        add (v_i, v_j) to matching M
return M
```

One graph class where this can be done is the *interval graphs*, i.e., the intersection graphs of horizontal segments in the plane. It was shown by Moitra and Johnson [26] that the greedy-algorithm always finds a maximum matching in an interval graph as long as we sort the vertices by left endpoint of their intervals. This gives an $O(m+n)$ algorithm for interval graphs since an interval representation can be found in $O(m+n)$ time [3]. Liang and Rhee [18] improve this further (presuming an interval representation is given) by rephrasing the greedy-algorithm as follows (see also Algorithm 2). Rather than adding an edge to the matching when the left endpoint (i.e., the one with the smaller index) is encountered, we add it when the right endpoint is encountered. We also explicitly maintain a data structure F that stores the *free* vertices, by which we mean vertices that were processed already but are as-of-yet unmatched. Liang and Rhee [18] implement F with a balanced binary search tree (storing left endpoints of intervals); then all operations required on F can be performed in $O(\log n)$ time. This therefore leads to an $O(n \log n)$ time algorithm for solving MATCHING in interval graphs; in particular this is *sub-linear* run-time if the graph has $\omega(n \log n)$ edges. This runtime can be easily improved to $O(n \log \log n)$ by using a van Emde Boas tree [31], as observed by Liang and Rhee [30].

Our results. In this paper, we take inspiration from [18] and develop sub-linear algorithms for MATCHING in *RDV graphs*, i.e., graphs that can be represented as vertex-intersection graphs of downward paths in a rooted tree T. (This is called an *RDV representation*;

*David R. Cheriton School of Computer Science, University of Waterloo, biedl@uwaterloo.ca

†David R. Cheriton School of Computer Science, University of Waterloo, prashant.gokhale@uwaterloo.ca

Algorithm 2: Delayed-Greedy-algorithm

Input: A graph G with a vertex order $v_1, \ldots, v_n$
Initialize the matching $M = \emptyset$
Initialize the set of free vertices $F = \emptyset$
for $j = 1, \ldots, n$ **do**
  if v_j *has neighbours in* F **then**
    among such neighbours, let v_i be the one that minimizes i
    add (v_i, v_j) to matching M
    delete v_i from F
  else
    add v_j to F
return M

formal definitions will be given below.) RDV graphs were introduced by Gavril [13]; many properties have been discovered and for many problems efficient algorithms have been found for RDV graphs [1, 20, 21, 29], quite frequently in contrast to only slightly bigger graph classes where the problem turns out to be hard. It is easy to see that all interval graphs are RDV graphs, so our results re-prove the results for interval graphs from [18].

RDV graphs can be recognized in polynomial time, and along the way an RDV representation is produced [13]. (The run-time has been improved, and even a linear-time algorithm has been claimed but without published details; see [4, Section 2.1.4] for more on the history.)

We show in this paper that if we are given an n-vertex graph G with an RDV representation on a tree T, then we can find a maximum matching in $O(|T| + n \log n)$ time. There always exists an RDV representation of G with $|T| \in O(n)$, so if we are given a suitable one then the run-time becomes $O(n \log n)$, hence sub-linear.

Our idea is to use the delayed-greedy-algorithm (Algorithm 2), and to pick a suitable data structure for the set F of free vertices. The key ingredient here is that 'does v_j have a neighbour in F' can be re-phrased, using the RDV representation, as the question whether a vertical segment intersects an element of a dynamically changing set of horizontal line segments, and if so, to return the one with maximal y-coordinate among them. This question in turn can be phrased as an orthogonal ray-shooting query, for which suitable data structures are known to exist. The current best implementation of them uses linear space and $O(\log n)$ time per operation [14]; this gives our result since we need $O(n)$ operations. We use the ray-shooting data structure as a black box, so if the run-time were improved (e.g. one could dream of $O(\log \log n)$ run-time if coordinates are integers in $O(n)$, as they are in our application) then the run-time of our matching-algorithm would likewise improve. Finally, we also study some possible improvements and extensions.

Other related results: There are a number of other results concerning fast algorithms to solve Matching in intersection graphs of some geometric objects. The results for interval graphs were extended to *circular arc graphs* (intersection graphs of arcs of a circle) [18]. In an entirely different approach, Matching can also be solved very efficiently in *permutation graphs* (intersection graphs of line segments connecting two parallel lines) [30]; see also [9] for an (unpublished) matching-algorithm for permutation graphs that is slower but beautifully uses range queries to find the matching. We should note that RDV graphs are unrelated to circular arc graphs and permutation graphs (i.e., neither a subclass nor a superclass); Figure 1 gives a specific example. As such, these results do not directly impact ours or vice versa.

Permutation graphs are a special case of *co-comparability graphs*, i.e., graphs for which the complement has an acyclic transitive orientation; these can also be viewed as intersections of curves between two parallel lines [15]. For these, maximum matchings can be found in linear time [24].

Finally, the greedy-algorithm actually works beyond interval graphs; in particular Dahlhaus and Karpinski [8] showed that it finds the maximum matching for *strongly chordal graphs*. These are the graphs that are *chordal* (every cycle C of length at least 4 has a *chord*, i.e., an edge between two non-consecutive vertices of C), and where additionally every even-length cycle C has a chord (v, w) such that an odd number of edges of C lie between v and w. The question whether chordal graphs have a linear-time algorithm for Matching remains open. But likely the answer is no, because as argued in [8], a linear-time algorithm for testing the existence of a *perfect matching* (i.e., a matching of size $n/2$) in a chordal graph would imply a linear-time algorithm for the same problem in any bipartite graph that is *dense* (has $\Theta(n^2)$ edges).

Graph Class	Runtime	Reference
Interval graphs	$O(n \log n)$	[18]
Interval graphs	$O(n \log \log n)$	Section **??**
Circular arc graphs	$O(n \log n)$	[18]
Permutation graphs	$O(n \log \log n)$	[30]
Strongly chordal graphs	$O(n + m)$	[8]
Co-comparability graphs	$O(n + m)$	[24]
RDV graphs	$O(n \log n)$	Section 3

Table 1: Existing and new results for Matching in some classes of graphs, presuming a suitable intersection representation is given and sufficiently small.

Table 1 gives an overview of existing and new results

for MATCHING in some classes of intersection graphs of objects. Our paper is structured as follows. After reviewing some background in Section 2, we give our main result for RDV graphs in Section 3. We briefly discuss interval graphs, as well as other possible extensions and open problems in Section 4.

2 Background

In this paper we study vertex-intersection graphs of subtrees of trees. We first define this formally, and then restrict the attention to a specific subclass.

Definition 1 *Let G be a graph. A* representation of G as a vertex-intersection graph of subtrees of a tree *consists of a* host-tree T *and, for each vertex v of G, a subtree $T(v)$ of T such that (v, w) is an edge of G if and only if $T(v)$ and $T(w)$ share at least one node of T.*

Such an intersection representation is sometimes also called a *clique-tree*, and slightly abusing notation, we use the word clique-tree also for the host tree T where convenient. As convention, we use the term 'node' for the vertices of the clique-tree, to distinguish them from the vertices of the graph represented by it. It is well-known that a graph has a clique-tree if and only if it is chordal [12]. We now review some properties of clique-trees that have been rooted.

Definition 2 *Let G be a graph with a rooted clique-tree T. For any vertex v, let $t(v)$ be the topmost (closest to the root) vertex in the subtree $T(v)$ of v. A* bottom-up enumeration *of G is a vertex order obtained by sorting vertices by decreasing distance of $t(v)$ to the root, breaking ties arbitrarily.*

Note that this bottom-up enumeration can be computed in $O(|T| + n)$ time, presuming every vertex v stores a reference to $t(v)$.

It will be convenient to assign points in $\mathbb{R}^2$ to the nodes of clique-tree T as follows. First, fix an arbitrary order of children at each node, and then enumerate the leaves of T as $L_1, \ldots, L_\ell$ from left to right. For every node i in T, let $\ell(i)$ be the leftmost (i.e., lowest-indexed) leaf that is a descendant of i, and set $x(i)$ to be the index of $\ell(i)$. We also need the notation $r(i)$ for the rightmost leaf that is a descendant of i. Also, define $y(i)$ to be the distance of node i from the root of the clique-tree. Figure 1 shows each node i drawn at point $(x(i), y(i))$ (where y-coordinates increase top-to-bottom). We can compute $x(\cdot)$ with a post-order traversal and $y(\cdot)$ with a BFS-traversal of host-tree T in $O(|T|)$ time.

RDV graphs and friends: Numerous subclasses of chordal graphs can be defined by studying graphs that have a clique-trees with particular properties. Most prominent here is the idea to require that $T(v)$ is a path. This gives the *path graphs* (also known as *VPT graphs*). One can further restrict the paths to be directed (after imposing some edge-directions onto the clique-tree); these are the *directed path graphs*. One can restrict this even further by requiring that the directions of the clique-tree are obtained by rooting the clique-tree, and this is the graph class that we study.

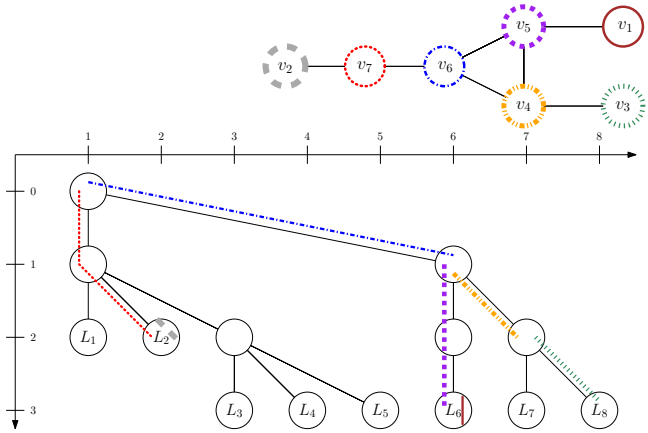


Figure 1: An RDV graph together with one possible RDV representation (for illustrative purposes the clique-tree is much bigger than needed). Nodes are drawn at their coordinates, and vertices are enumerated in bottom-up order. The graph is neither a circular arc graph nor a permutation graph.

Definition 3 *A* rooted directed path graph *(or* RDV graph *[27]) is a graph that has an* RDV representation*, i.e., a clique-tree that has been rooted and for every vertex v the subtree $T(v)$ is a* downward path*, i.e., a path that begins at some node and then always goes downwards.*

See Figure 1 for an example of an RDV representation.[1]

3 Matching in RDV graphs

Assume for the rest of this section that we are given an RDV representation of a graph G. In what follows, we will often use '$P(v)$' in place of '$T(v)$' for the subtree of a vertex v, to help us remind ourselves that these are downward paths rather than arbitrary trees. Recall that $t(v)$ denotes the top (closest to the root) node of $P(v)$; because we have a downward path (rather than an arbitrary tree) representing v we can now also define $b(v)$ to be the bottom node of $P(v)$.

[1] 'RDV' comes from 'rooted directed vertex-intersection'. Gavril called these 'directed path graphs' [13], but this later got used for the more general graphs where the directions need not be obtained via rooting.

For run-time purposes we presume that 'the RDV representation is given' means that we have a rooted tree T and (for each vertex v of G) two references $b(v)$ and $t(v)$ to the nodes of T that define the downward path.

Farber [10] showed that RDV graphs are strongly chordal, and Dahlhaus and Karpinski [8] showed that the greedy algorithm works correctly on strongly chordal graphs if we consider vertices in a so-called *strong elimination order* (which is usually assumed to be given with a strongly chordal graph). This suggests that the greedy-algorithm works for RDV graphs, but there is one missing piece: How do we get a strong elimination order from an RDV representation efficiently? This is very easy (use the bottom-up enumeration), and the proof that it works is not hard, but requires some more definitions and is therefore delayed to the appendix.

Theorem 1 *Let G be a graph with a given RDV representation. Then the greedy matching algorithm, applied to a bottom-up enumeration, returns a maximum matching.*

Exactly as in [18], to achieve a sub-linear run-time we will not use the greedy-algorithm directly but instead use the equivalent delayed-greedy-algorithm (Algorithm 2). The main bottleneck for the run-time is then to implement a data structure for the set F of free vertices. Such a data structure should store indexed vertices and must support the following three operations:

A. insert a new vertex

B. delete a vertex

C. query for the smallest neighbour, i.e., given a vertex v_j not in F, either determine that v_j has no neighbours in F, or return the neighbour v_i of v_j in F that minimizes index i.

The first two operations are straightforward, but the third one is non-trivial if we want to use $o(degree(v_j))$ time. To this end, we reduce adjacency queries in an RDV graph to the question of whether a horizontal segment intersects a vertical segment. We need some definitions first.

Definition 4 *Let G be a graph with an RDV representation. For each vertex v, define the following (see Figure 2 for examples):*

- *The* horizontal segment $s(v)$ *of v is the segment between the point of $t(v)$ and $(x(r(t(v))), y(t(v)))$, i.e., it extends rightward until it is above the rightmost descendant of $t(v)$.*
- *The* vertical segment $q(v)$ *of v is the segment between the point of $b(v)$ and $(x(b(v)), y(t(v)))$, i.e., it extends upward until it is to the right of $t(v)$.*

Recall that $t(v)$ has the same x-coordinate as its leftmost descendant, so the x-range of segment $s(v)$ is exactly the range of x-coordinates among descendants of $t(v)$. We also note that the name 'q' for the vertical segment was chosen since this will be used to implement the query operation.

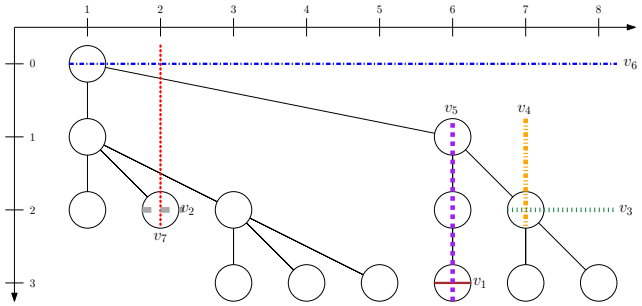


Figure 2: Mapping the vertices of the example in Figure 1 to horizontal and vertical segments (not all are shown).

Theorem 2 *Let G be a graph with an RDV representation and let $v_1, \ldots, v_n$ be the bottom-up enumeration of vertices. Then for any $i < j$, edge (v_i, v_j) exists if and only if the vertical segment $q(v_j)$ intersects the horizontal segment $s(v_i)$.*

Proof. Since $q(v_j)$ is a vertical segment and $s(v_i)$ is a horizontal segment, they intersect if and only if both the x-coordinates and y-coordinates line up correctly, i.e., $x(t(v_i))$=$x(\ell(t(v_i))) \leq x(b(v_j)) \leq x(r(t(v_i)))$ and $y(t(v_j)) \leq y(t(v_i)) \leq y(b(v_j))$.

Assume first that edge (v_i, v_j) exists, which means that $P(v_i)$ and $P(v_j)$ have a node u in common. Among all such nodes u, pick the one that is closest to the root; this implies $u \in \{t(v_i), t(v_j)\}$. By $i < j$ we actually know $u = t(v_i)$, because if $u \neq t(v_i)$ then u=$t(v_j)$ would be a strict descendant of $t(v_i)$ and have larger y-coordinate, contradicting the bottom-up elimination ordering. Since $u \in P(v_j)$, node $b(v_j)$ is a descendant of u (which in turn is a descendant of $t(v_j)$), so $y(t(v_j)) \leq y(u)$=$y(t(v_i)) \leq y(b(v_j))$ and the y-coordinates line up. The x-coordinates line up since $b(v_j)$ is a descendant of $t(v_i)$=u and the horizontal segment $s(v_i)$ covers all such descendants.

Assume now that the segments intersect. By $y(t(v_j)) \leq y(t(v_i)) \leq y(b(v_j))$ then path $P(v_j)$ contains a node (call it u) with $y(u) = y(t(v_i))$. If u equals $t(v_i)$ then $P(v_i)$ and $P(v_j)$ have node $t(v_i)$ in common and (v_i, v_j) is an edge as desired. If $u \neq t(v_i)$, then these two nodes (with the same y-coordinate) have a disjoint set of descendants, so the intervals $I_u = [x(\ell(u)), x(r(u))]$ and $I_i = [x(\ell(t(v_i))), x(r(t(v_i)))]$ are disjoint. Since $b(v_j)$ is a descendant of $u \in P_j$, we have $x(b(v_j)) \in I_u$, but since

the x-coordinates line up we have $x(b(v_j)) \in I_i$. This is impossible. □

In light of this insight, we now can reformulate our requirements on a data structure for F as follows. We want to store horizontal segments (associated with vertices of a graph) and must be able to support the following three operations:

A'. insert a new horizontal segment,

B'. delete a horizontal segment,

C'. query whether a vertical segment $q(v_j)$ intersects a segment in F, and if so, among all intersected segments return the segment $s(v_i)$ that maximizes the y-coordinate.

We can reformulate C' as a ray-shooting query as follows. Replace the vertical segment $q(v_j)$ by a vertical ray $\vec{q}(v_j)$ obtained by directing $q(v_j)$ upward. (So the ray originates at the point of $b(v)$ and goes vertically towards smaller y-coordinates.)

Observation 1 *To perform operation C′, it suffices to do the following:*

C". query whether a ray $\vec{q}(v_j)$ intersects a segment in F, and if so, among all intersected segments return the first segment $s(v_i)$ that is hit by the ray.

Proof. At the time of the query, the set F of free vertices contains only segments of vertices v_i with $i < j$. Therefore all segments intersected by ray $\vec{q}(v_j)$ have y-coordinate at least $y(t(v_j))$, and also intersect the segment $q(v_j)$. So we will only report correct segments. Since the ray is vertically upward (while y-coordinates increase in downward direction), the first segment that is hit is the one that maximizes the y-coordinate. □

Operation C" is the well-known *orthogonal ray-shooting* problem, and operations A' and B' means that we want a dynamic variant. Many data structures have been developed for this (some for more general versions), see for example [23], [17] for older results with slower processing time. For the orthogonal ray shooting problem specifically, the best run-time bounds achieved are by Giyora and Kaplan [14], who showed how to implement all three operations in $O(\log n)$ time, using $O(n)$ space (assuming the data structure stores up to n items). Later on this was generalized to drop the requirement of orthogonality [28] without affecting space or runtime. Some of these data structures assume that the line segments are disjoint. The horizontal segments we have defined earlier are not necessarily disjoint, but we can make them disjoint (without affecting the outcome) by adding $\frac{n-i}{n}$ to the y-coordinate of $s(v_i)$.

With this, we can put everything together into our main theorem.

Theorem 3 *Given an n-vertex graph G with an RDV representation T, the maximum matching of G can be found in $O(|T| + n \log n)$ time.*

Proof. Parse T to compute the x-coordinates and y-coordinates of all nodes in T, then bucket-sort the vertices by decreasing $y(t(v))$ to obtain the bottom-up order $v_1, \ldots, v_n$ in $O(|T| + n)$ time. By Theorem 1 applying the greedy-algorithm with this vertex-ordering will give a maximum matching. Using the delayed greedy-algorithm, the run-time of the algorithm is reduced to performing operations A-C $O(n)$ times. By storing the free set F as horizontal segments, this by Theorem 2 is the same as performing operations A', B' and C" $O(n)$ times. Using a suitable data structure for orthogonal ray shooting [14], this takes $O(\log n)$ time per operation and hence $O(n \log n)$ time in total. □

One can easily argue that any RDV graph has an RDV representation T with $|T| \in O(n)$, for otherwise two adjacent nodes of T are used by the same set of subtrees and could be combined into one. So the run-time becomes $O(n \log n)$ if a suitably small RDV representation is given.

Recall that for interval graphs, an improvement of the run-time for matching from $O(n \log n)$ to $O(n \log \log n)$ is possible by exploiting that all intervals can be described via integers in $O(n)$ and storing F using van Emde Boas trees [30]. This naturally raises an open question: Could the run-time of Theorem 3 also be improved to $O(n \log \log n)$ time, presuming $|T| \in O(n)$? The bottleneck for this would be to improve the run-time for the orthogonal ray-shooting data structure if all coordinates are (small) integers. This question was explicitly asked by Giyora and Kaplan [14], and appears to be still open. Could we at least achieve run-time $O(n(\log \log n)^k)$ for some constant k for RDV graphs?

4 Clique trees where subtrees have few leaves

An RDV graph is a chordal graph with a rooted clique-tree where every subtree $T(v)$ has exactly one leaf. A natural generalization of this graph class are the chordal graphs with a rooted clique-tree where every subtree $T(v)$ has at most Δ leaves. (A very similar concept was introduced by Chaplick and Stacho under the name of *vertex leafage* [5]; the only difference is that they considered unrooted clique trees and so count the root of $T(v)$ as leaf if it has degree 1.)

Theorem 4 *Let G be a graph with a rooted clique-tree T where all subtrees have at most Δ leaves. Then the greedy-algorithm applied to the bottom-up enumeration can be implemented in $O(|T| + \Delta n \log n)$ time.*

Proof. For each vertex v, split $T(v)$ into $k \leq \Delta$ paths $P_1(v), \ldots, P_k(v)$, each connecting the root $t(v)$ of $T(v)$

to a leaf, such that their union covers all of $T(v)$. Define segment $s(v)$ as before (it only depends on $t(v)$), and define k query-segments $q_1(v), \ldots, q_k(v)$ for the paths. One easily verifies that for $i < j$ vertex v_j is a neighbour of v_i if and only if at least one of $q_1(v_j), \ldots, q_k(v_j)$ intersects $s(v_i)$. So to perform operation C, we do a ray-shooting query for each of $\vec{q_1}(v_j), \ldots, \vec{q_k}(v_j)$ and choose among the returned segments (if any) the one that has maximum y-coordinate. With this operation C can be implemented in $O(\Delta \log n)$ time. All other aspects of the greedy-algorithm are exactly as in Section 3. □

Unfortunately, this does not improve the time to find maximum matchings for such graphs, because there is no guarantee that the greedy-algorithm finds a maximum matching when applied with a bottom-up enumeration. To see a specific example, consider the *directed path graphs* (recall that these are obtained by requiring $T(v)$ to be a directed path after directing the clique-tree, but the edge-directions need not come from rooting the clique-tree). This is a strict superclass of RDV graphs, for example the graph in Figure 3, which is also known as *4-trampoline*, is a directed path graph but not an RDV graph since it is not even strongly chordal [11]. For any choice of root, every path $T(v)$ becomes a subtree with at most two leaves, and so the greedy-algorithm can be implemented in $O(n \log n)$ time (presuming the clique-tree was small). Unfortunately, this does not necessarily give a maximum matching, see Figure 3.

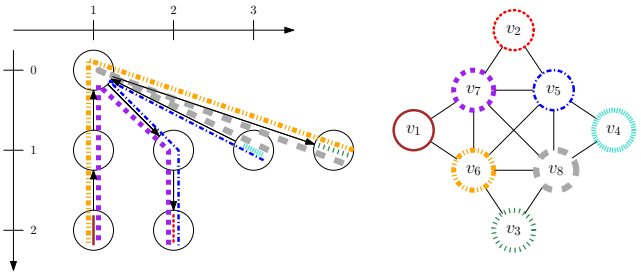


Figure 3: A directed path graph that is not strongly chordal. With the depicted bottom-up enumeration, the greedy-algorithm would choose matching (v_5, v_2), (v_6, v_1), (v_8, v_3) and leave v_4, v_7 unmatched even though the graph has a matching of size 4.

This raises another natural open problem: Can we find a maximum matching in a directed path graph (with a given small clique-tree) in $O(n \log n)$ time? How about the path graphs, an even broader class?

References

[1] L. Babel, I. Ponomarenko, and G. Tinhofer. The isomorphism problem for directed path graphs and for rooted directed path graphs. *Journal of Algorithms*, 21(3):542–564, 1996.

[2] C. Berge. *Graphs and Hypergraphs, 2nd edition*. North-Holland, 1976. Translated from *Graphes et Hypergraphes, Dunod, 1970*.

[3] K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *Journal of Computing and System Sciences*, 13:335–379, 1976.

[4] S. Chaplick. PQR-trees and undirected path graphs. Master's thesis, Department of Computer Science, University of Toronto, 2008. `https://hdl.handle.net/1807/118597`.

[5] S. Chaplick and J. Stacho. The vertex leafage of chordal graphs. *Discrete Applied Mathematics*, 168:14–25, 2014.

[6] L. Chen, R. Kyng, Y. Liu, R. Peng, M. Probst Gutenberg, and S. Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022.

[7] J. Chuzhoy and S. Khanna. A faster combinatorial algorithm for maximum bipartite matching. *CoRR*, abs/2312.12584, 2023.

[8] E. Dahlhaus and M. Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1):79–91, 1998.

[9] F. Bauernöppel and E. Kranakis and D. Krizanc and A. Maheshwari and J.-R. Sack and J. Urrutia. An improved maximum matching algorithm in a permutation graph, 1995. Manuscript. Improved version of Technical Report TR-95-06, School of Computer Science, Carleton University, Ottawa, Canada.

[10] M. Farber. *Applications of L.P. duality to problems involving independence and domination*. PhD thesis, Rutgers University, 1982.

[11] M. Farber. Characterizations of strongly chordal graphs. *Discret. Math.*, 43(2-3):173–189, 1983.

[12] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

[13] F. Gavril. A recognition algorithm for the intersection graphs of directed paths in directed trees. *Discret. Math.*, 13(3):237–249, 1975.

[14] Y. Giyora and H. Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Trans. Algorithms*, 5(3):28:1–28:51, 2009.

[15] M. Golumbic, D. Rotem, and J. Urrutia. Comparability graphs and intersection graphs. *Discrete Mathematics*, 43(1):37–46, 1983.

[16] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New York, 1st edition, 1980.

[17] H. Kaplan, E. Molad, and R. Tarjan. Dynamic rectangular intersection with priorities. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 639–648, New York, NY, USA, 2003. Association for Computing Machinery.

[18] Y. Liang and C. Rhee. Finding a maximum matching in a circular-arc graph. *Information Processing Letters*, 45(4):185–190, 1993.

[19] M. C. Lin and J. L. Szwarcfiter. Characterizations and recognition of circular-arc graphs and subclasses: A survey. *Discret. Math.*, 309(18):5618–5635, 2009.

[20] M.-S. Lin and S.-H. Su. Counting maximal independent sets in directed path graphs. *Information Processing Letters*, 114(10):568–572, 2014.

[21] M.-S. Lin and C.-C. Ting. Computing the k-terminal reliability of directed path graphs. *Information Processing Letters*, 115(10):773–778, 2015.

[22] L. Lovász and M. D. Plummer. *Matching theory*, volume 29 of *Annals of Discrete Mathematics*. North-Holland Publishing Co., Amsterdam, 1986.

[23] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5(1–4):215–241, 2023.

[24] G. Mertzios, A. Nichterlein, and R. Niedermeier. A linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *SIAM J. Discret. Math.*, 32(4):2820–2835, 2018.

[25] S. Micali and V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 17–27, 1980.

[26] A. Moitra and R. Johnson. A parallel algorithm for maximum matching on interval graphs. In *Proceedings of the International Conference on Parallel Processing, ICPP '89*, pages 114–120. Penn State University Press, 1989.

[27] C. Monma and V. Wei. Intersection graphs of paths in a tree. *J. Comb. Theory, Ser. B*, 41(2):141–181, 1986.

[28] Y. Nekrich. Dynamic planar point location in optimal time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 1003–1014, New York, NY, USA, 2021. Association for Computing Machinery.

[29] C. Papadopoulos and S. Tzimas. Computing a minimum subset feedback vertex set on chordal graphs parameterized by leafage. *Algorithmica*, 86(3):874–906, 2024.

[30] C. Rhee and Y. Liang. Finding a maximum matching in a permutation graph. *Acta Informatica*, 32(8):779–792, 1995.

[31] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical systems theory*, 10:99–127, 1976.

[32] V. V. Vazirani. A proof of the MV matching algorithm. *CoRR*, abs/2012.03582, 2020.

Appendix

5 Proof of Theorem 1

To prove the theorem, we first need some definitions. Write $N[v]$ for the *closed neighbourhood* of a vertex, i.e. the set consisting of v and all its neighbours. Call a vertex v *simple* [11] if $N[v]$ is a clique that can be ordered as $w_1, \ldots, w_k$ such that $N[w_1] \subseteq N[w_2] \subseteq \cdots \subseteq N[w_k]$. The crucial ingredient is the following observation:

Lemma 5 *Let G be a graph with an RDV representation. Then a vertex v_1 that maximizes $y(t(v_1))$ is simple.*

Proof. Since v_1 maximizes $y(t(v_1))$, node $t(v_1)$ must belong to $P(w)$ for any neighbour w of v. This shows immediately that $N[v_1]$ is a clique since all subtrees of neighbours share $t(v_1)$.

Now remove all nodes from the RDV representation that have y-coordinate strictly bigger than $y(t(v_1))$; by choice of v_1 this does not remove any adjacencies. If we now sort the neighbours of v as $w_1, \ldots, w_k$ by decreasing y-coordinate of their top endpoints, then (since all subtrees are downward paths that end at $t(v_1)$) we have $P(w_1) \subseteq \cdots \subseteq P(w_k)$ and so v_1 is simple. □

We can view a bottom-up elimination order $v_1, \ldots, v_n$ as using a vertex v that maximizes $y(t(v))$ as v_1, removing it from the graph, and repeating until the graph is empty. By the above, then each v_i is a simple vertex with respect to the graph induced by $\{v_i, \ldots, v_n\}$. Farber [11, Theorem 3.3] showed that a vertex order with this property is a strong elimination order, and as mentioned earlier, using a strong elimination order guarantees that the greedy-algorithm for matching succeeds [8].

6 The graph of Figure 1

We claimed earlier that the graph G in Figure 1 is neither a circular arc graph nor a permutation graph, and we briefly argue this here. We repeat the graph here for convenience.

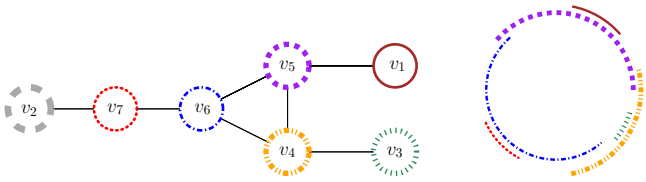


Figure 4: The graph G of Figure 1 and a circular arc representation of $G \setminus \{v_2\}$.

Most of our argument considers only the graph $G \setminus \{v_2\}$. This is well-known not to be a comparability graph [16, Figure 5.1], and since permutation graphs are subgraphs of comparability graphs and closed under vertex-deletion, G is not a permutation graph.

Next observe that vertices $\{v_1, v_3, v_7\}$ form what is known as an *asteroidal triple*: any two of them can be connected via a path that avoids the neighbourhood of the third. No such structure can exist in an interval graph. In fact, $G \setminus$

$\{v_2\}$ is known to be an obstruction for *Helly circular-arc graphs* [19], i.e., it does not have a circular arc representation where for every clique C the arcs of vertices in C all share a common point. Since $\{v_4, v_5, v_6\}$ is the only non-trivial clique, therefore in any circular arc representation of $G \setminus \{v_2\}$ the three arcs of v_4, v_5, v_6 do not share a common point. To still have pairwise intersections, these three arcs together cover the entirety of the circle. But then we cannot add an arc for v_2 anywhere since it has no edge to any of these three vertices. Therefore G is not a circular arc graph.

Polyhedral roll-connected colorings of partial tilings

Robert D. Barish* Tetsuo Shibuya†

Abstract

We consider the problem of coloring the faces of an edge-to-edge partial tiling $\mathcal{T}$ of the xy-plane in such a manner that a face-colored polyhedron $\mathcal{P}$ "rolling" over this tiling – where, using less precision for the moment, each time a face of the polyhedron is superimposed on a congruent tile in $\mathcal{T}$, both the face and the tile must have the same coloring – can reach any tile from any other tile. Here, for $\mathcal{P}$ corresponding to any Platonic solid, we show that the existence of such a coloring with at most $w \geq 1$ distinct colors can be decided in $\mathcal{O}(\mathcal{T})$ time. On the other hand, when we require at least two internally disjoint manners of rolling from any starting location to any ending location, and when $\mathcal{P}$ is the cube and $w = 3$, we show that deciding the existence of and counting such colorings becomes NP-hard and #P-hard, respectively.

1 Introduction

In rolling polyhedra puzzles and mazes, which appear to trace their origins to the mathematician Roland Sprague [24] (see also Harris [19]), the challenge is traditionally to roll a polyhedron such as a *Platonic cube* (i.e., a cube with the symmetries of a Platonic solid) over a sequence of its edges and towards some objective. This objective may, for example, be that it arrives at a particular destination with some specified end state orientation, or that it traces a Hamiltonian tour of some subset of cells on a board, in each case with potential complications in the form of obstacles or positions which can only be visited when the polyhedron has a certain orientation. While a full review of the history of these puzzles and mazes is outside the scope of the current work (see, e.g., Buchin et al. [8] for a very thorough overview), we remark that such geometric games have become a staple of recreational mathematics, and have been highlighted in multiple instances of Martin Gardner's "Mathematical Games" column between 1963 and 1975 [13, 16, 17] (with solutions to problems presented each subsequent month [12, 14, 15]). We also refer the reader to the work of Baes et al. [2] for an in-depth analysis and history of rollings of various polyhedra on more general tessellations.

Perhaps due to the combination of these games having a simple intuitive description, as well as at times unexpected depth and difficulty, there exists a substantial literature to date on the computational complexity of solving rolling polyhedra puzzles and mazes (see, e.g., [2, 7, 8, 20, 28]). We also briefly remark that reachability and coverage problems for rolling polyhedra have been studied in a mechanical engineering context as a nice example of a system under discrete nonholonomic (i.e., path dependent) constraints [4, 21].

In this work, based on these games, and inspired in part by the notion of rainbow connected colorings [9, 10] and proper connected colorings [1, 6] of graphs, we introduce and analyze a "Menger-like" notion of connectivity [22] for face colorings of polygons composing edge-to-edge partial tilings of the xy-plane, where these tilings are defined as follows:

Definition 1. *Edge-to-edge partial tiling.* Letting $\mathcal{S}$ be a set of polygonal tiles embedded in the xy-plane, we refer to $\mathcal{S}$ as an edge-to-edge partial tiling if no two polygons share a common interior point (i.e., polygons are not permitted to overlap) and any adjacent polygons have exactly one full edge in common (implying vertices for this common edge must also overlap).

Here, in lieu of considering internally vertex disjoint simple paths between all pairs of vertices in a graph, we consider internally disjoint rollings of polyhedra – per Definition 2 (given below) – between all pairs of face colored polygons in edge-to-edge partial tilings.

Definition 2. *Colored polyhedral rolling.* Letting $\mathcal{P}$ be a polyhedron with a face coloring $\mathcal{H}_{\mathcal{P}}$, letting $\mathcal{T}$ correspond to an edge-to-edge partial tiling of the xy-plane, and letting $\mathcal{C}_{\mathcal{T}}$ be a tuple specifying face colorings for the polygons in $\mathcal{T}$, a colored polyhedral rolling is a continuous rigid body motion of $\mathcal{P}$ on $\mathcal{T}$ satisfying the following constraints:

- (Motion Constraint 1) all interior points of $\mathcal{P}$ must maintain a positive non-zero z-component;
- (Motion Constraint 2) in the state where a face of

*Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan, rbarish@ims.u-tokyo.ac.jp

†Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan, tshibuya@hgc.jp

This work was supported by JSPS Kakenhi grants {23H03345, 23K18501, 20H05967, 21H05052}.

$\mathcal{P}$ is superimposed on a congruent polygonal tile in $\mathcal{T}$, the cube can only be moved by being rotated around an edge shared with $\mathcal{T}$ until another face of $\mathcal{P}$ is superimposed on another congruent polygonal tile in $\mathcal{T}$;

- (Motion Constraint 3) any superimposed face of $\mathcal{P}$ and polygon in $\mathcal{T}$ must be identically colored by $\mathcal{H}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{T}}$, respectively;
- (Motion Constraint 4) the initial and final states for the motion must have a face of $\mathcal{P}$ superimposed on a congruent polygon of $\mathcal{T}$.

To give a higher level and more intuitive description of Definition 2, (Motion Constraint 1) is a "hard surface" requirement saying that the polyhedron must roll "on top" of the tiling $\mathcal{T}$ rather than below or through it, and (Motion Constraint 2) is saying that $\mathcal{P}$ is moved by being "tipped over" some succession of its edges, or in other words, rotated about an edge on the edge-to-edge partial tiling of the xy-plane until such time that a new face of $\mathcal{P}$ is superimposed on a new congruent polygonal tile in $\mathcal{T}$. Next, (Motion Constraint 3) is saying that any face of $\mathcal{P}$ superimposed on a polygonal tile in $\mathcal{T}$ must have the same coloring as this tile. Finally, (Motion Constraint 4) expresses that we must begin and end the rolling motion with $\mathcal{P}$ having a face superimposed on a polygonal tile in $\mathcal{T}$.

For visual intuition concerning the rigid body motion allowed under (Motion Constraint 1) through (Motion Constraint 4), we refer the reader to the Figure 1 for an illustration of a Platonic cube, with a distinct coloring for each face, rolling on an edge-to-edge partial tiling. As a further clarification, we remark that $\mathcal{P}$ can "revisit" any polygon in $\mathcal{T}$ any number of times, and in any orientation so long as (Motion Constraint 1) through (Motion Constraint 4) are satisfied.

Provided this context, we can now define a disjointness notion for colored polyhedral rollings by the following Definition 3, and our "Menger-like" notion of roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k)$-connected w-colorings by the following Definition 4:

Definition 3 *Internally disjoint colored polyhedral rollings.* Letting X and Y be two colored polyhedral rollings for a polyhedron $\mathcal{P}$ on an edge-to-edge partial tiling $\mathcal{T}$, letting F_X and F_Y be the set of polygons in $\mathcal{T}$ that are superimposed on some congruent face of $\mathcal{P}$ over the course of X and Y, respectively, and letting $\{f_{(X,initial)}, f_{(X,final)}\} \subseteq F_X$ and $\{f_{(Y,initial)}, f_{(Y,final)}\} \subseteq F_Y$ be the initial and final states for X and Y, respectively, we say that X and Y are internally disjoint colored polyhedral rollings if and only if $\big(F_X \setminus \{f_{(X,initial)}, f_{(X,final)}\}\big) \cap \big(F_Y \setminus \{f_{(Y,initial)}, f_{(Y,final)}\}\big) = \emptyset$.

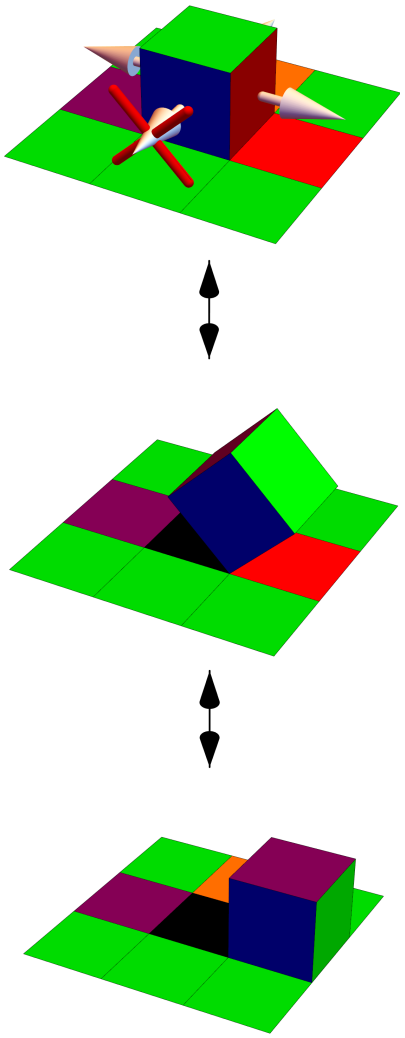


Figure 1: Illustration of a Platonic cube $\mathcal{P}$, with all unique face colorings, undergoing a rigid body "tipping over" motion under (Motion Constraint 1) through (Motion Constraint 4) – see Definition 2 – on a set of colored squares composing an edge-to-edge partial tiling of the plane; **(top)** initial stage in which the black face of $\mathcal{P}$ is superimposed on a congruent black colored square of the edge-to-edge partial tiling, and (white) arrows show possible and forbidden (indicated via a red "X") directions in which $\mathcal{P}$ may be "tipped over"; **(middle)** intermediate stage of a "tipping over" motion for $\mathcal{P}$; **(bottom)** final stage of a "tipping over" motion for $\mathcal{P}$, in which the red face of $\mathcal{P}$ is superimposed on a congruent red colored square of the edge-to-edge partial tiling.

Definition 4. *Roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k)$-connected w-coloring.* Assuming the definitions and notation from Definition 2, an assignment $\mathcal{C}_{\mathcal{T}}$ of at most $w \in \mathbb{N}_{>0}$ distinct colors to the polygons in $\mathcal{T}$ is a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k)$-connected w-coloring if it ensures the existence of $k \in \mathbb{N}_{>0}$ inter-

nally disjoint colored polyhedral rollings of $\mathcal{P}$ between all pairs of polygons in $\mathcal{T}$.

To elaborate on Definition 3 and Definition 4 at a higher level of abstraction, consider the "dual graph" $D_{\mathcal{T}}$ for an edge-to-edge partial tiling $\mathcal{T}$ generated by placing a vertex at the center of each polygonal tile and adding an edge between a pair of vertices whenever two polygonal tiles are adjacent. Next, think of a polyhedral rolling on $\mathcal{T}$ (under the constraints from Definition 2) as a walk on $D_{\mathcal{T}}$, where we say that the walk visits a given vertex whenever a face of $\mathcal{P}$ is superimposed on a congruent polygonal tile corresponding to that vertex in $\mathcal{T}$. In this context, we can understand the notion of internally disjoint colored polyhedral rollings from Definition 3 as corresponding to internally vertex disjoint walks between the same pair of vertices in $D_{\mathcal{T}}$. Furthermore, we can understand the notion of a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k)$-connected w-coloring from Definition 4 as a coloring of the faces of $\mathcal{T}$ (equiv., coloring of the vertices in $D_{\mathcal{T}}$) using at most w colors, where for a polyhedron $\mathcal{P}$ with a face coloring specified by $\mathcal{H}_{\mathcal{P}}$, there exist at least k internally disjoint colored polyhedral rollings for $\mathcal{P}$ (equiv., at least k internally vertex disjoint walks in $D_{\mathcal{T}}$) between all pairs of polygonal tiles in $\mathcal{T}$ (equiv., all vertex pairs in $D_{\mathcal{T}}$).

To briefly outline the structure of this paper, in the proceeding Section 2, we analyze the state transitions that can occur for the Platonic cube with two different types of face colorings during a colored polyhedral rolling. Subsequently, in Section 3 we establish that deciding the existence of a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k)$-connected w-coloring is tractable in polynomial time (with respect to the size of the edge-to-edge partial tiling) in the case where $k = 1$ (Proposition 1). However, we also show that the problem becomes NP-hard in the case where $k = 2$, even when $\mathcal{P}$ corresponds to the Platonic cube with a face coloring $\mathcal{H}_{\mathcal{P}}$ identically coloring all opposing pairs of faces using $w = 3$ distinct colors (Theorem 1). Finally, in Section 4, after first showing that the problem of counting proper 3-colorings of a planar graph of degree at most 4 is #P-complete under many-one counting reductions (Theorem 2), we subsequently establish as a corollary that the counting version of the decision problem from Theorem 2 is #P-hard under many-one counting reductions (Corollary 1).

2 State transition graphs for the face-colored Platonic cube rolling on edge-to-edge partial tilings

To represent the full state space for a polyhedron with colored faces rolling on an edge-to-edge partial tiling under (Motion Constraint 1) through (Motion Constraint 4) from Definition 2, we define the following:

Definition 5. *State transition graph.* Assuming the definitions and notation from Definition 2, for a fixed polyhedron $\mathcal{P}$ and a fixed face coloring $\mathcal{H}_{\mathcal{P}}$ for $\mathcal{P}$, a state transition graph is constructed by: (1) creating a vertex for every possible manner in which a face of $\mathcal{P}$ may be superimposed on a congruent polygon in an edge-to-edge partial tiling; (2) making a pair of vertices v_a and v_b adjacent if and only if a colored polyhedral rolling can possibly exist where $\mathcal{P}$ can transition from state v_a to v_b by rotating about (i.e., being "tipped over") exactly one edge in an edge-to-edge partial tiling.

Here, for two explicit examples of state transition graphs, we can consider:

- the Platonic cube with all distinct face colorings;
- the Platonic cube where $w = 3$ distinct colors are used to identically color each opposing pair of faces.

Concerning the former example where $\mathcal{P}$ corresponds to the Platonic cube with all distinct face colorings, in Figure 2(a) we show the planar "unfolding" of this instance of $\mathcal{P}$, in which vertices on each polyhedron face are labeled in accordance with their corresponding vertices in the folded polyhedron. Next, in Figure 2(b), we give an illustration of our notation for the state transition graph shown in Figure 2(c), where the square polygon represents the "bottom" face of $\mathcal{P}$ currently superimposed on a congruent polygon in an edge-to-edge partial tiling, "N", "W", "E", and "S" indicate North, West, East, and South motion (i.e., "tipping over") directions, and we represent the state of the cube with the tuple (A, B, C, D) of labels for the vertices of the "bottom" face in the Southwest (i.e., A), Southeast (i.e., B), Northwest (i.e., C), and Northeast (i.e., D) corners. Finally, in Figure 2(c), we show the state transition graph for $\mathcal{P}$, in which vertices are again labeled in accordance with the tuple notation from Figure 2(b) as well as colored in accordance with the face specified by the vertex labels, (solid) arrows show transitions resulting from motion in the Northward direction, and (dotted) arrows show transitions resulting from motion in the Eastward direction.

Concerning the latter example where $\mathcal{P}$ corresponds to the Platonic cube with all opposing faces identically colored using $w = 3$ distinct colors, Figure 3(a) shows the planar "unfolding" of this instance of $\mathcal{P}$, in which vertices on each polyhedron face are again labeled in accordance with their corresponding vertices in the folded polyhedron. Next, in Figure 3(b), we show the state transition graph for $\mathcal{P}$, following the notation shown in Figure 2(b,c), with the exception that prefixes appended to each tuple of labels for the vertices of the "bottom" face (e.g., "R1", "R2", "B1", "B2") indicate equivalent states of $\mathcal{P}$ given its face colorings.

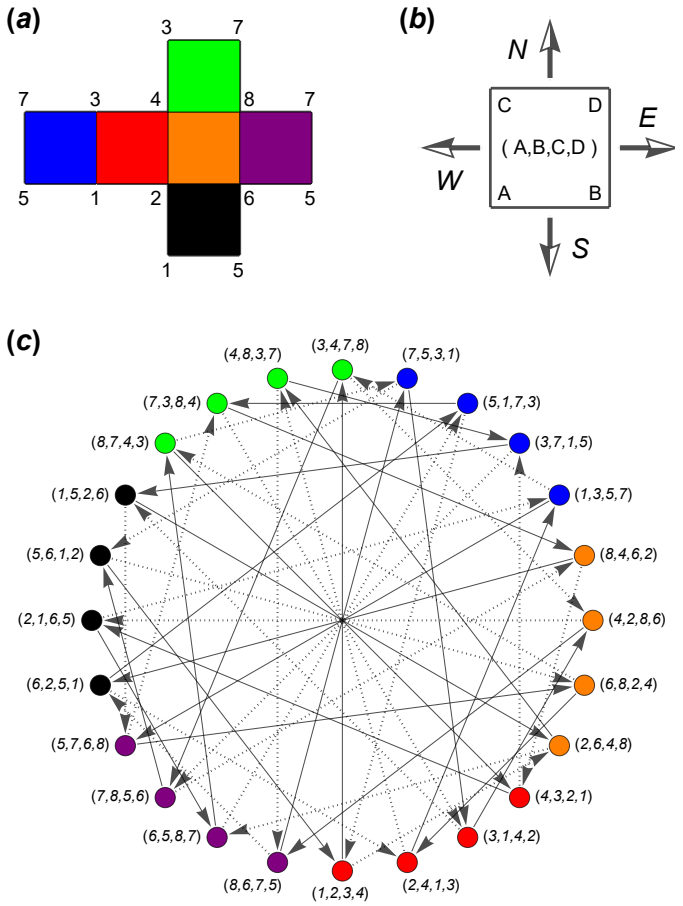


Figure 2: Analysis of the state transitions for a Platonic cube $\mathcal{P}$, imbued with all unique face colorings, undergoing a colored polyhedral rolling; **(a)** planar "unfolding" of $\mathcal{P}$; **(b)** illustration showing our notation for the state transition graph; **(c)** state transition graph for $\mathcal{P}$; see Section 2 for further details.

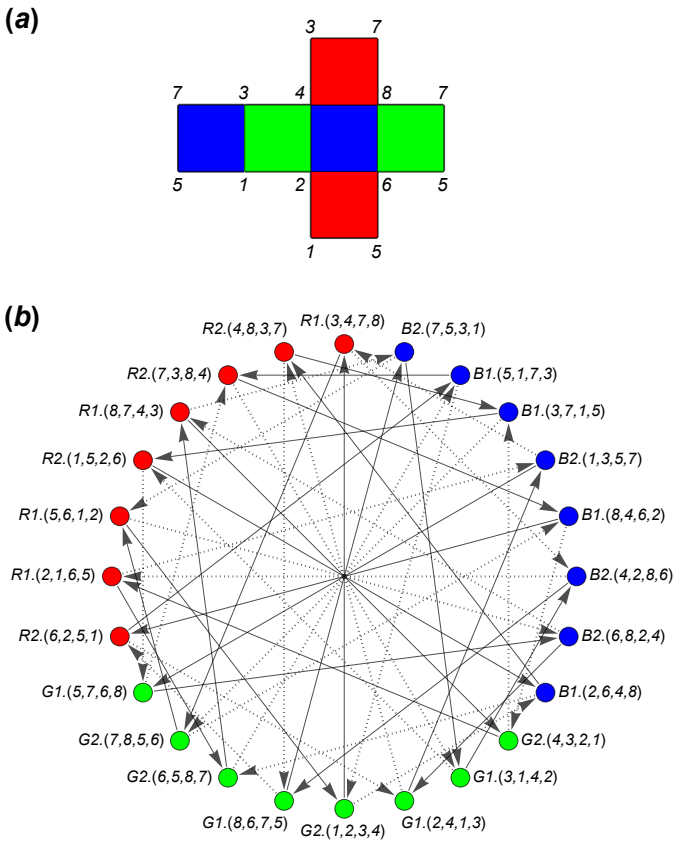


Figure 3: Analysis of the state transitions for a Platonic cube $\mathcal{P}$, in which each opposing pair of faces is assigned a distinct coloring, undergoing a colored polyhedral rolling; **(a)** planar "unfolding" of $\mathcal{P}$; **(b)** state transition graph for $\mathcal{P}$; see Section 2 for further details.

3 The complexity of finding and deciding the existence of some roll-connected colorings

For the following Proposition 1 and Theorem 1, we follow the definitions and notation for $\mathcal{P}$, $\mathcal{H}_{\mathcal{P}}$, and $\mathcal{T}$ from Definition 2.

Proposition 1 *For fixed $\mathcal{P}$ corresponding to a Platonic solid, fixed $\mathcal{H}_{\mathcal{P}}$, and $w \in \mathbb{N}_{>0}$ corresponding to the number of unique face colors encoded by $\mathcal{H}_{\mathcal{P}}$, a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 1)$-connected w-coloring for an edge-to-edge partial tiling of the plane $\mathcal{T}$ can be found, or determined not to exist, in $\mathcal{O}(|\mathcal{T}|)$ time.*

Proof. Let $\mathcal{D}_{\mathcal{T}}$ be the dual graph for $\mathcal{T}$, generated by associating a vertex with each polygon and connecting a pair of vertices when their polygons share all points along an edge. Observe that $\mathcal{D}_{\mathcal{T}}$ will have at most $\mathcal{O}(|\mathcal{T}|)$ vertices and edges, as polygons in $\mathcal{T}$ correspond to an edge-to-edge partial tiling of a plane. Accordingly, we can perform a breadth-first or depth-first search in $\mathcal{O}(|\mathcal{T}|)$ time to check that $\mathcal{D}_{\mathcal{T}}$ is connected. If so, we can determine a minimum spanning tree $Q_{\mathcal{D}}$ for $\mathcal{D}_{\mathcal{T}}$, and if not, we can trivially determine that no roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 1)$-connected w-coloring can exist.

To proceed in the case where $Q_{\mathcal{D}}$ does exist, we begin by selecting an arbitrary root node v_r in $Q_{\mathcal{D}}$. Next, under (Motion Constraint 1) through (Motion Constraint 4), for any initial superimposition of the polygon corresponding to v_r with a congruent face of $\mathcal{P}$ (note that this initial choice doesn't matter, as $\mathcal{P}$ has the symmetries of a Platonic solid), we greedily color the polygons in $\mathcal{T}$ along each root to leaf path in $Q_{\mathcal{D}}$ to ensure that each such path corresponds to a colored polyhedral rolling for $\mathcal{P}$ with face colorings given by $\mathcal{H}_{\mathcal{P}}$. It remains to observe that this coloring of the polygons in $\mathcal{T}$ will necessarily be a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 1)$-connected w-coloring, with $w \in \mathbb{N}_{>0}$ again corresponding to the number of unique face colors encoded by $\mathcal{H}_{\mathcal{P}}$, and that all of the

aforementioned steps will take at most $\mathcal{O}(|\mathcal{T}|)$ time. □

Theorem 1 *For $\mathcal{P}$ corresponding to a Platonic cube, and $\mathcal{H}_{\mathcal{P}}$ identically coloring each opposing pair of faces of $\mathcal{P}$ using $w = 3$ distinct colors, it is NP-hard to decide the existence of a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 2)$-connected $(w = 3)$-coloring.*

Proof. Letting $\mathcal{P}$ correspond to Platonic cube with face colorings $\mathcal{H}_{\mathcal{P}}$ corresponding to the planar "unfolding" shown in Figure 3(a), we proceed via reduction from the NP-complete proper 3-coloring decision problem for planar 2-connected graphs of degree at most 4 [18].

To begin, for an arbitrary planar 2-connected graph of degree at most 4 denoted G, we will draw G on a $\mathbb{Z}^2$ integer lattice in such a manner that: (Requirement 1) all vertices of G fall on lattice points; (Requirement 2) all edges between vertices in G are encoded as non-crossing "polylines" composed of a series of end-to-end connected horizontal and vertical unit length segments; and (Requirement 3) all non-crossing "polylines" contain at least one vertical unit length segment. Concerning (Requirement 1) and (Requirement 2), we refer to such a drawing as an orthogonal integer lattice embedding, and due to G being a planar graph of degree at most 4, these drawings can be computed in time polynomial in the size of G via a number of well-known methods (see, e.g., [5, 23, 25, 26, 27]). For an explicit example of such an embedding, see Figure 4(a) for an instance of a planar graph having degree at most 4, and its (non-unique) orthogonal integer lattice embedding in Figure 4(b). Concerning (Requirement 3), in Figure 4(c) we show how it is possible to enlarge any such orthogonal integer lattice embedding via multiplying all embedding coordinates by some integer $k \in \mathbb{N}_{>0}$, and Figure 4(d) we show how a $(k = 5)$-fold enlargement of an orthogonal integer lattice embedding allows us to redraw "polylines" such that they contain at least one vertical unit segment.

Now let M_G be the orthogonal integer lattice embedding of G, let X be the set of vertical and horizontal unit length segments for polylines in M_G, and let $Y \subseteq X$ be a subset consisting of exactly one arbitrarily chosen vertical unit length segment for each polyline in M_G corresponding to an edge in G. We construct an edge-to-edge partial tiling of a plane $\mathcal{T}$ via the following steps: (step 1) we identify each vertical segment in $X \setminus Y$ with an appropriately scaled instance of the set of 15 polygons composing the "vertical color copying gadget" shown in Figure 5; (step 2) we identify each horizontal segment in $X \setminus Y$ with an appropriately scaled instance of the set of 15 polygons composing a "horizontal color copying gadget" shown in Figure 6, where this gadget is simply a $\frac{\pi}{2}$ rotation of the "vertical color copying gadget"; and (step 3) we identify each vertical segment in Y with an appropriately scaled instance of the set of 27 polygons composing the "vertical color change gadget" shown in Figure 7. Here, in each case, gadgets identified with adjacent unit length segments along a "polyline" will share a common "input / output" cell at the former intersection point of the unit segments, and we indicate such "input / output" cells with arrows in Figure 5, Figure 6, and Figure 7.

Noting that all possible roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 2)$-connected $(w = 3)$-colorings for the "vertical color copying gadget", "horizontal color copying gadget", and "vertical color change gadget" are shown in Figure 5, Figure 6, and Figure 7, respectively, that the "horizontal color copying gadget" is just a $\frac{\pi}{2}$ rotation of the "vertical color copying gadget", and observing the state transition graph from Figure 3(b), we now make the following observations:

- (obs. 1) any colored polyhedral rolling over the polygons in a "vertical color copying gadget" or "horizontal color copying gadget" will map $\mathcal{P}$ to an identical orientation on an identically colored polygon of $\mathcal{T}$ (e.g., $R1.(\ldots) \rightarrow R1.(\ldots)$ or $B2.(\ldots) \rightarrow B2.(\ldots)$);

- (obs. 2) for $\mathcal{P}$ with initial configuration $R1.(\ldots)$, $B1.(\ldots)$, or $G1.(\ldots)$ (respectively, $R2.(\ldots)$, $B2.(\ldots)$, or $G2.(\ldots)$), rolling over the polygons in a "vertical color change gadget" can map $\mathcal{P}$ to any orientation in the set $(R1.(\ldots), B1.(\ldots), G1.(\ldots))$ (respectively, $(R2.(\ldots), B2.(\ldots), G2.(\ldots))$) under the constraint that the beginning and ending polygons in the edge-to-edge partial tiling must have distinct colors;

- (obs. 3) by brute force enumeration and explicitly checking each possible square tile coloring of the "vertical color copying gadget", "horizontal color copying gadget", and "vertical color change gadget" (which has $3^{27} = 7625597484987$ possible colorings when not accounting for symmetries), in the context of a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 2)$-connected $(w = 3)$-coloring, only the colorings of the gadgets shown in Figure 5, Figure 6, and Figure 7 are possible, and this holds even if the input graph G is only 2-connected.

Putting everything together, by (obs. 1) through (obs. 3) we can determine that a roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 2)$-connected $(w = 3)$-coloring can exist for $\mathcal{T}$ only if G admits a proper 3-coloring. Noting that G is 2-connected, we can also determine that such a coloring will always exist if G is proper 3-colorable. This yields the current theorem. □

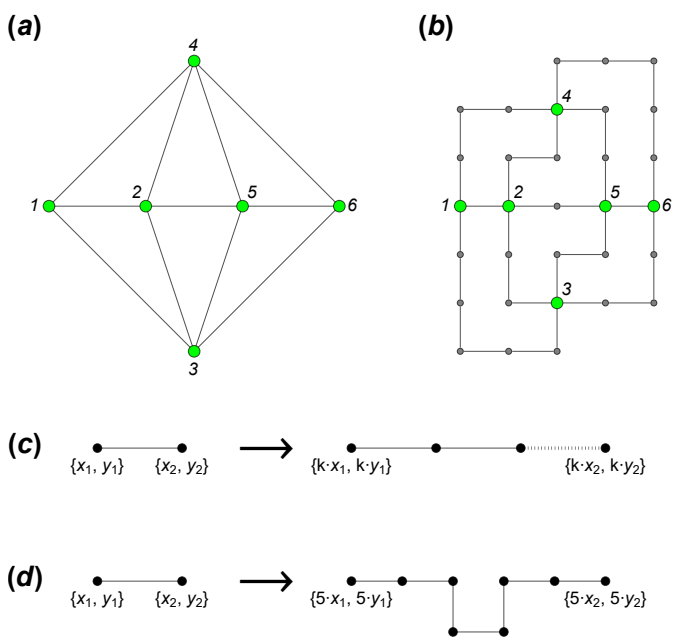


Figure 4: An example orthogonal $\mathbb{Z}^2$ integer lattice embedding; **(a)** planar undirected graph on 6 (labeled) vertices with vertex degree at most 4; **(b)** orthogonal integer lattice embedding of the graph shown in (a), in which all edges have unit length, (larger black) vertices are labeled in accordance with the vertices they correspond to in (a), and (smaller gray) vertices indicate the vertical and horizontal unit length edge segments composing each non-crossing "polyline"; **(c)** illustration of how an orthogonal integer lattice embedding can be expanded or "blown up" by moving all embedded points on the grid from their coordinate $\{x, y\}$ to the coordinate $\{k \cdot x, k \cdot y\}$ for some $k \in \mathbb{N}_{>0}$; **(d)** illustration of how a $(k = 5)$-fold expansion of an orthogonal integer lattice embedding allows one to ensure that all "polylines" contain at least one vertical unit length segment.

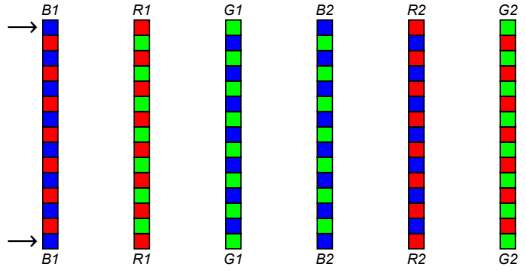


Figure 5: All possible roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 2)$-connected $(w = 3)$-colorings of the "vertical color copying gadget" when $\mathcal{P}$ and $\mathcal{H}_{\mathcal{P}}$ correspond to the Platonic cube with the Figure 3(a) planar "unfolding"; for the left-most gadget instance, arrows indicate "input / output" cells; note that the "horizontal color copying gadget" is simply a $\frac{\pi}{2}$ rotation of the "vertical color copying gadget".

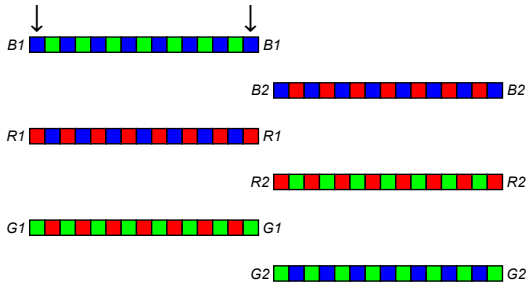


Figure 6: All possible roll-$(\mathcal{P}, \mathcal{H}_{\mathcal{P}}, k = 2)$-connected $(w = 3)$-colorings of the "horizontal color copying gadget" when $\mathcal{P}$ and $\mathcal{H}_{\mathcal{P}}$ correspond to the Platonic cube with the Figure 3(a) planar "unfolding"; for the top-left-most instance of the gadget, arrows indicate the "input / output" cells; note that the "vertical color copying gadget" is simply a $\frac{\pi}{2}$ rotation of the "horizontal color copying gadget".

4 The complexity of counting some roll-connected colorings

Theorem 2 *Counting proper 3-colorings of a planar graph of degree at most 4 is #P-complete under many-one counting reductions.*

Proof. Observe that counting proper 3-colorings of an arbitrary planar graph is #P-complete under many-one counting reductions [3]. Here, letting G be such a planar graph, our task will be to convert G into a planar graph of degree at most 4, denoted G', and to do so while preserving the number of proper 3-colorings of G.

To begin, we borrow a planar 4-regular "3-valent color copying gadget" originally due to Dailey [11], corresponding to the subgraph induced by the (green) vertices in Figure 8, which has the property that the "output" vertices labeled v_a, v_b, and v_c must have the same coloring in any proper 3-coloring. Using this gadget, it is straightforward to construct a planar degree ≤ 4 "k-valent color copying gadget", with k "output" vertices of degree 2 having the same coloring and belonging to the same face of the gadget. For instance, to construct the "k-valent color copying gadget" shown in Figure 8, one can take k copies of the "3-valent color copying gadget", and for some cyclic ordering of these gadgets, identify the v_b vertex of each gadget with the v_a vertex of the subsequent gadget in the ordering.

Now, letting S be the set of all vertices in G having degree ≥ 5, and letting $deg\,(v)$ be the degree of a vertex v, we can construct G' by replacing all vertices $v \in S$ in G with a "$deg\,(v)$-valent color copying gadget". In particular, we can substitute each edge with an endpoint at $v \in S$ with an edge having an endpoint at a distinct instance of one of the "output" vertices, and preserve planarity by doing so in a manner that respects

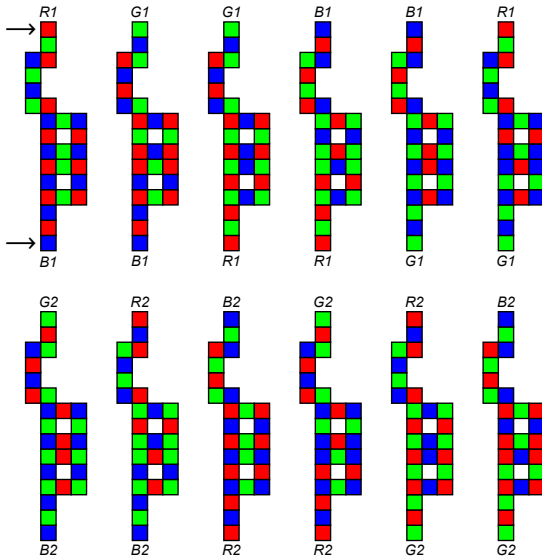


Figure 7: All possible roll-$(\mathcal{P}, \mathcal{H}_\mathcal{P}, k = 2)$-connected $(w = 3)$-colorings of the "vertical color change gadget" when $\mathcal{P}$ and $\mathcal{H}_\mathcal{P}$ correspond to the Platonic cube with the Figure 3(a) planar "unfolding"; for the top-left-most gadget instance, arrows indicate "input / output" cells.

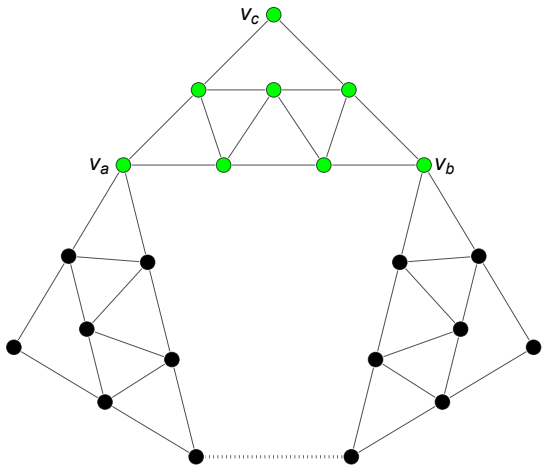


Figure 8: Illustration of the planar 4-regular "k-valent color copying gadget" used in the Theorem 3 proof argument, where the subgraph induced by the (green) vertices is originally due to Dailey [11].

the rotational system for the embedding of G. Finally, using brute force enumeration to verify that the original "3-valent color copying gadget" due to Dailey [11] has exactly 2 proper 3-colorings per coloring of the vertex labeled v_a in Figure 8 (implying 6 proper 3-colorings overall), we can determine that there will be $\prod_{v \in S} 2^{deg(v)}$ proper 3-colorings for G' per proper 3-coloring for G.

Putting everything together, as the problem of counting proper 3-colorings of planar degree ≤ 4 graphs is straightforwardly in #P, and as many-one counting reductions are transitive, this yields the current theorem. □

Corollary 1 *For $\mathcal{P}$ corresponding to a Platonic cube, and $\mathcal{H}_\mathcal{P}$ identically coloring each opposing pair of faces of $\mathcal{P}$ using $w = 3$ distinct colors, it is #P-hard to count roll-$(\mathcal{P}, \mathcal{H}_\mathcal{P}, k = 2)$-connected $(w = 3)$-colorings.*

Proof. Recall the Theorem 1 proof argument, in which we gave a reduction from deciding the existence of a proper 3-coloring for a planar 2-connected graph G of degree at most 4 to deciding the existence of a roll-$(\mathcal{P}, \mathcal{H}_\mathcal{P}, k = 2)$-connected $(w = 3)$-coloring for the same specifications of $\mathcal{P}$ and $\mathcal{H}_\mathcal{P}$. Here, by Theorem 2 and (obs. 3) of the Theorem 1 proof argument, it suffices to observe that, per proper 3-coloring of G, there will be exactly 2 possible roll-$(\mathcal{P}, \mathcal{H}_\mathcal{P}, k = 2)$-connected $(w = 3)$-colorings for the edge-to-edge partial tiling constructed from G. □

5 Concluding remarks and open problems

Our initial objective was to classify the complexity of finding and counting roll-$(\mathcal{P}, \mathcal{H}_\mathcal{P}, k = 2)$-connected w-colorings for all five Platonic solids, and in each case, for all possible non-equivalent face colorings. However, as evidenced by the reduced scope of the current work, this proved much harder and more tedious than expected. Accordingly, we challenge the reader to come up with an efficient manner of moving forward with this classification effort, in particular, one which allows us to more easily recognize computationally tractable cases such as a Platonic octahedron in which $w = 2$ colors are used to ensure that no pair of adjacent faces have the same coloring. In addition, noting that we were only able to establish an NP-hardness (as opposed to NP-completeness result) with our Theorem 1 proof argument, a further challenge is to come up with an efficient algorithm, should one exist, for verifying that a particular face coloring of an edge-to-edge partial tiling admits a roll-$(\mathcal{P}, \mathcal{H}_\mathcal{P}, k = 2)$-connected w-coloring.

More broadly, we point out that it is possible to consider higher dimension colored polyhedral rollings, where for instance one can permit the rolling polyhedron to rotate to the "underside" of a tiling embedded in $\mathbb{R}^3$. We argue that such generalizations may be interesting for at least the reason that, in cases where $k \geq 3$, it is no longer trivial to decide the existence of roll-$(\mathcal{P}, \mathcal{H}_\mathcal{P}, k)$-connected w-colorings. Finally, and regardless of the dimension of a particular tiling, we strongly suspect that the counting and decision problems considered in this work will be fixed-parameter tractable for a parameter based of the number of holes in a given tiling, and leave the proof of this as an open problem.

References

[1] E. Andrews, C. Lumduanhom, E. Laforge, and P. Zhang. On proper-path colorings in graphs. *J. Comb. Math. Comb. Comput.*, 97:189–207, 2016.

[2] A. Baes, E. D. Demaine, M. L. Demaine, E. Hartung, S. Langerman, J. O'Rourke, R. Uehara, Y. Uno, and A. Williams. Rolling polyhedra on tessellations. *Proc. 11th international conference on Fun with Algorithms (FUN)*, pages 5:1–5:16, 2022.

[3] R. Barbanchon. On unique graph 3-colorability and parsimonious reductions in the plane. *Theoret. Comput. Sci.*, 319(1–3):455–482, 2004.

[4] A. Bicchi, Y. Chitour, and A. Marigo. Reachability and steering of rolling polyhedra: a case study in discrete nonholonomy. *IEEE Trans. Autom. Control.*, 49(5):710–726, 2004.

[5] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom.*, 9(3):159–180, 1998.

[6] V. Borozan, S. Fujita, A. Gerek, C. Magnant, Y. Manoussakis, L. Montero, and Z. Tuza. Proper connection of graphs. *Discrete Math.*, 312(17):2550–2560, 2012.

[7] K. Buchin and M. Buchin. Rolling block mazes are PSPACE-complete. *J. Inf. Process.*, 20(3):719–722, 2012.

[8] K. Buchin, M. Buchin, E. D. Demaine, M. L. Demaine, D. El-Khechen, S. P. Fekete, C. Knauer, A. Schulz, and P. Taslakian. On rolling cube puzzles. *Proc. 19th annual Canadian Conference on Computational Geometry (CCCG)*, pages 141–144, 2007.

[9] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang. Rainbow connection in graphs. *Math. Bohem.*, 133(1):85–98, 2008.

[10] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang. The rainbow connectivity of a graph. *Networks*, 54(2):75–81, 2009.

[11] D. P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Math.*, 30(3):289–293, 1980.

[12] M. Gardner. Mathematical games. *Scientific American*, 232(4):126–133, April 1975.

[13] M. Gardner. Mathematical games. *Scientific American*, 209(6):140–148, December 1963.

[14] M. Gardner. Mathematical games. *Scientific American*, 213(6):100–105, December 1965.

[15] M. Gardner. Mathematical games. *Scientific American*, 210(1):120–128, January 1964.

[16] M. Gardner. Mathematical games. *Scientific American*, 232(3):112–116, March 1975.

[17] M. Gardner. Mathematical games. *Scientific American*, 213(5):116–123, November 1965.

[18] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976.

[19] J. Harris. Single vacancy rolling cube problems. *J. Recreat. Math.*, 7(3):220–224, 1974.

[20] M. Holzer and S. Jakobi. On the complexity of rolling block and Alice mazes. *Proc. 6th international conference on Fun with Algorithms (FUN)*, pages 210–222, 2012.

[21] A. Marigo, Y. Chitour, and A. Bicchi. Manipulation of polyhedral parts by rolling. *Proc. 1997 IEEE International Conference on Robotics and Automation (IRCA)*, 4:2992–2997, 1997.

[22] K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[23] A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Comput. Geom.*, 9(1–2):83–110, 1998.

[24] R. Sprague. *Unterhaltsame mathematik: neue probleme – überraschende lösungen (Problem 3: Schwere Kisten)*. Friedr. Vieweg & Sohn, Verlag: Braunschweig, 1961.

[25] J. A. Storer. On minimal-node-cost planar embeddings. *Networks*, 14:181–212, 1984.

[26] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[27] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, 36(9):1230–1234, 1989.

[28] A. Uejima and T. Okada. NP-completeness of rolling dice puzzles using octahedral and icosahedral dices. *IEICE Trans. Fundamentals (Japanese Edition)*, A94(8):621–628, 2011.

Slant/Gokigen Naname is NP-complete, and Some Variations are in P

Jayson Lynch* Jack Spalding-Jamieson†

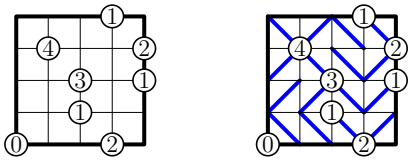


Figure 1: A simple example of a Slant puzzle, and its (unique) solution.

Abstract

In this paper we show that a generalized version of the Nikoli puzzle Slant is NP-complete. We also give polynomial time algorithms for versions of the puzzle where some constraints are omitted. These problems correspond to simultaneously satisfying connectivity and vertex degree constraints in a grid graph and its dual.

1 Introduction

Gokigen Naname, also known as Slant, is a pencil-and-paper logic puzzle by the Publisher Nikoli[1]. The puzzle involves filling every square in a grid with diagonal lines so they do not create cycles and each circle has the indicated number of diagonals touching it. See Fig. 1 for an example of a Slant puzzle.

Rules Slant is played on a square grid, some of whose vertices may be given numbers (henceforth called a ***Slant board***). The objective is to add a single diagonal line to each square obeying the following constraints:

1. There is no cycle formed by diagonal lines.
2. If a vertex has a designated number $k \in \{0, 1, 2, 3, 4\}$, the number of diagonal lines adjacent to the vertex must be exactly k.

These shall henceforth be referred to as the ***cycle constraint*** and ***vertex constraints***. Examples of solutions violating each can be seen in Fig. 2.

Results In Section 2 we discuss some combinatorial properties of the puzzle which will be used in our algorithmic and computational complexity results.

In Section 3 we give multiple algorithmic results for special cases of the Slant problem. We show that Slant is fixed-parameter tractable in the number of vertex constraints in the puzzle. We give an algorithm for deciding if a partially filled board can be completed without violating the cycle constraint. Finally we show how to formulate the vertex constraints and the cycle constraint as a matroid intersection problem allowing us to solve instances of the puzzle if the vertex constraints stay within certain partitions of the vertices.

Finally, in Section 4, we show that solving an instance of Slant on an $n \times m$ grid is NP-complete via a reduction from Hamiltonian cycle in max-degree-3 planar graphs.

Related Work Many pencil-and-paper puzzles have been studied from the lens of computational complexity, especially those by the designer Nikoli. For example the following puzzles all have results showing generalized versions to be NP-complete: Angle Loop [30], Bag / Corral [11], Chained Block [20], Country Road [15], Fillomino [34], Five-Cells[18], Hashiwokakero [7], Hebi [21], Heyawake [14], Hiroimono / Goishi Hiroi [6], Kouchoku [30], Kurodoko [23], LITS [27], Masyu / Pearl [12], Mid-Loop [30], Nagareru Loop [19], Nagenawa [30], Nurimeizu[19], Numberlink [24], Nurikabe [26, 13], Satogaeri [21], Shakashaka [10, 3], Slitherlink [35, 34, 2], Suraromu [21], Tatamibari [4], Yajilin [15], and Yosenabe [17]. There have also been multiple surveys on the topic of computational complexity and games and puzzles[32, 22, 8].

Previous work showed NP-hardness for Yin-Yang puzzles and connected it to the problem of partitioning vertices into two trees with some vertices being preassigned[9]. We find that Slant is also equivalent to a fairly natural graph problem in which we want to partition edges between a graph and its dual such that each forms a single tree with some vertices having degree constraints. This is discussed in more detail in Section 2.

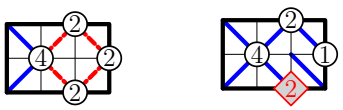


Figure 2: An example of each class of constraint violation denoted in red (the cycle constraint (left) and a vertex constraint (right)).

*MIT-CSAIL, jaysonl@mit.edu
†David R. Cheriton School of Computer Science, University of Waterloo, jacksj@uwaterloo.ca

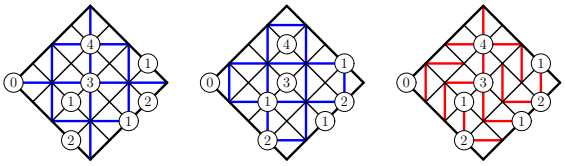


Figure 3: An example of how the Slant puzzle from Fig. 1 can be rotated 45° so that all its potential edges come from two complementary grid graphs.

2 Preliminary Results and Key Observations

We discuss some preliminary results about Slant that will be used in other parts of the paper.

First, we observe that the two choices of diagonals have a parity constraint. Each possible diagonal connects two vertices which differ by exactly one in both their x and y-coordinates. Thus, we can partition the grid into points whose coordinate-sum $(x + y)$ is even, and points whose coordinate-sum is odd. Every square then admits a diagonal connecting two odd-sum points or a diagonal connecting two even-sum points. Furthermore, if we rotate all of our coordinates by 45 degrees it becomes obvious that every such diagonal belongs to one of two square grids graphs (see Fig. 3). Moreover, the two square grid graphs are essentially planar duals of each other. That is, the faces of one correspond to the vertices of the other, and vice versa, with the same edge-vertex and edge-face incidences (with some exceptions for the boundary). Thus, choosing a diagonal involves deciding to place an edge either in a primal square grid graph or a dual square grid graph.

With this dual graph perspective, we can see that the acyclic constraint also implies that both graphs form forests where every tree touches the boundary. By adding edges connecting the various points at which the boundary is touched, the complementary forests can be turned into a tree-cotree structure. This also means that each parity class must contain roughly half the edges, in particular $(n-1)(m-1)/2+O(n+m)$ edges and the average degree is also $2 + O(\frac{n+m}{nm})$.

3 Polynomial Time Algorithms for Special Cases

We present variants of Slant that can be solved in polynomial time which use subclasses of the constraints. The most important result in this section is a positive result on partially-filled boards:

Theorem 1 *Given a partially-filled Slant board with no vertex constraints, whose filled-in diagonals do not already form a cycle, there always exists a valid solution extending the partially-filled in diagonals.*

Proof. Iterate through the unfilled squares in any order and make a choice of diagonal arbitrarily. If that choice induces a cycle, choose the other diagonal instead. These two choices of diagonals connect vertices of different parities (odd degree vs even degree coordinate-sums), and thus any curve they are a part of do not share vertices. Further, each pair of vertices is on opposite sides of the diagonal. Thus by required planarity of the connections and the Jordan curve theorem[1][31], the other diagonal will not induce a cycle (see Fig. 4). □

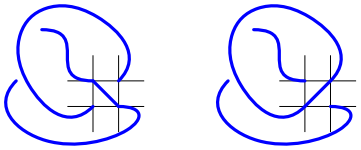


Figure 4: An example of why if one configuration of a square induces a cycle, the other cannot.

Theorem 1 will be particularly useful as a component of the NP-hardness result in Section 4, and we will also use it to prove results in this section.

Corollary 2 *Given a Slant board whose designated numbers are all either* 0 *or* 4*, it can be checked in polynomial time if the board permits a valid solution.*

Proof. Every vertex constraint whose designated number is 0 or 4 requires the same set of diagonals in any valid solution. In particular, degree-4 constrained vertices constraints require all four edges to be adjacent, and 0 vertex constraints can only appear on the boundary (otherwise they force a cycle) and all edges to not be adjacent. If there are any conflicts between these requirements, then the board does not permit a valid solution. Otherwise, generate the corresponding partially-filled Slant board. If the board contains a cycle, then it contains a cycle in any solution, so it does not permit a valid solution. Otherwise, apply Theorem 1 to obtain a valid extension to the partially-filled board. □

3.1 Dense and Sparse Representations

The decision variant of the Slant problem (whether or not a valid solution exists) is trivially in NP, but only if it is assumed that $\Theta(nm)$ values are used to represent the full $n \times m$ grid (the ***dense representation***). However, since not every vertex is required to have a designated number, one could instead represent an instance of Slant with only the grid size, and a list of non-empty coordinates with designated numbers (the ***sparse representation***). In this representation, the size of the input may be asymptotically much smaller than $n \times m$. We find that this does not change the complexity class:

Theorem 3 *For an $n \times m$ Slant board with k vertex constraints, using a sparse representation, it can be*

[1]Here we can use the simpler result that a simple polygonal curve divides the plane into two regions

checked if a valid solution exists in non-deterministic poly($k \log(n+m)$) time.

Proof. For a certificate, consider a sparse repsentation of a partially-filled Slant board whose filled squares are exactly those adjacent to any vertex with a designated number. After non-deterministically assigning values to such a certificate, it can be checked whether the designated numbers are satisfied, and whether there are any cycles present among the assigned diagonals in $O(k)$ time. If all designated numbers are satisfied by the partially-filled solution, and no cycles are present, by Theorem 1, there exists some valid extension of the partially-filled Slant grid, and hence the answer to the decision problem is "YES". □

This same argument also implies that Slant is Fixed-Parameter Tractable in the number of vertex constraints. As before, we can exhaustively enumerate all of the possible configurations around the k clues and then use Theorem 1 to check for a valid extension.

Corollary 4 *Slant can be solved in time $O(16^k(n+m))$ where k is the number of vertex clues, and is thus FPT in the number of vertex clues.*

3.2 Matroid Structure of Slant

In this subsection, we show that Slant can be formulated as the intersection of five matroids. While this on its own is not particularly surprising (Slant is clearly in NP, and matroid intersection for ≥ 3 matroids is known to be NP-hard [29]), the particular structure of the matroids will give us insight into some relaxations of Slant's constraints that are polynomially-time solvable. We will specifically show that Slant can be formulated as a weighted intersection of four partition matroids, and one planar matroid.

A partition matroid is induced by a partition $\{P_i\}_i$ of its ground set E, and a mapping $P_i \mapsto b_i \in \mathbb{Z}_{\geq 0}$ so that a set $S \subset E$ is independent if and only if $|S \cap P_i| \leq b_i$.

A planar matroid is one that is graphic and cographic. That is, its basis is the set of maximal forests in some graph (i.e., its independent sets are the forests in that graph), and the basis of its dual is also the set of maximal forests in some graph. Equivalently, the graph defining the graphic matroid is a planar graph.

First, we will observe that for an $n \times n$ Slant board, the vertex constraints alone (without the cycle constraint) can be solved with an $0-1$ integer linear programming problem. This will be done by creating a variable for each square indicating the choice of diagonal, and a constraint for each vertex with a designated number. The details of the construction are as follows:

- Use coordinates $(0,0)$ for both the top-left vertex of the grid, and for the top-left square of the grid.
- For each square with coordinates (x,y), create a variable $s_{x,y} \in \{0,1\}$.
- For each vertex with coordinates (x,y) whose designated number is $k \in \{0,1,2,3,4\}$, add an equality constraint: If $x+y$ is even, require that $\sum_{x_\Delta, y_\Delta \in \{-1,0\}} s_{x+x_\Delta, y+y_\Delta} = k$. Else, require that $\sum_{x_\Delta, y_\Delta \in \{-1,0\}} (1 - s_{x+x_\Delta, y+y_\Delta}) = k$. Equivalently, that $\sum_{x_\Delta, y_\Delta \in \{-1,0\}} (s_{x+x_\Delta, y+y_\Delta}) = 4-k$.

Given an assignment s satisfying these constraints, a corresponding solution to the Slant puzzle can be constructed as follows: For a square whose coordinates are (x,y), if $s_{x,y} = 1$, draw a diagonal between the two incident vertices whose coordinate-sums are even. Otherwise, draw a diagonal between the two incident vertices whose coordinate-sums are odd.

We can use this ILP formulation to also obtain a matroid intersection formulation. Let E be the set of cells. E will form the ground set for all five of our matroids. Essentially, we will consider subsets $S \subset E$, and map these to possible solutions to a Slant instance. A subset S will represent the set of edges connecting incident vertices with even coordinate-sums, and likewise the set $E \setminus S$ will represent the set of edges connecting incident vertices with odd coordinate-sums. In this sense, the variables $s_{x,y}$ in the ILP formulation can be thought of as a indicator functions for some subset $S \subset E$.

Consider the graph G formed by connecting all vertices with even coordinate-sums. This graph is bipartite, so call its parts A and B. A Slant instance induces a partition matroid over $E(G)$ for each of A and B, which we will denote as $\mathcal{L}_A$ and $\mathcal{L}_B$: The independent sets of $\mathcal{L}_A$ are the subsets of edges which do not surpass the specified values at the vertices in A. Similarly, the independent sets of $\mathcal{L}_B$ are the subsets of edges which do not surpass the specified values at the vertices in B. In this case, $E(G) = E$ as defined before, so these are partition matroids over E, as desired.

We can also form a graph G' formed by connecting all vertices with odd coordinate-sums. Call its parts C and D. We define partition matroids over $E(G')$ in a symmetric manner, except that in place of a specified value k at a vertex in C or D, we use $4-k$. Let the resulting partition matroids be denoted as $\mathcal{L}_C$ and $\mathcal{L}_D$.

With this formulation, the above ILP, encapsulating all but the cycle constraint, can be solved via a 4-way matroid intersection: If a set S is an independent set in all of $\mathcal{L}_A$, $\mathcal{L}_B$, $\mathcal{L}_C$, and $\mathcal{L}_D$, then its corresponding indicator s function does not violate relaxed constraints of the form: $\sum_{x_\Delta \in \{-1,0\}} \sum_{y_\Delta \in \{-1,0\}} s_{x+x_\Delta, y+y_\Delta} \leq k$ (resp. $4-k$, if applicable). We weight the elements of E by summing the left-hand side of all such constraints, and we let N be the sum of all right-hand sides. Let S' be an independent set in all four matroids, such that its indicator function s' is of maximum weight. Then, s' has total weight equal to N if and only if all such

constraints must be tight, and hence if and only if S' satisfies all non-cycle constraints of the Slant instance.

Lastly, we consider the cycle constraint, which was not encapsulated in the ILP formulation. A graph G is cycle-free if and only if it is a forest. This is encapsulated by a graphic matroid, whose base set is $E(G)$. In this case, the graph we consider is that of the even coordinate-sum vertices. This graph is planar and connected (in fact, it's a grid graph). Hence, its basis consists of trees with m edges for some value of m. Call this matroid $\mathcal{L}_N$. We modify the 4-way matroid intersection to be a 5-way matroid intersection as follows: Add 1 to the weight of every edge in E as determined for the 4-way intersection. Let S' be an independent set in all five matroids, such that its indicator function s' is of maximum weight. Then, s' has total weight equal to $N+m-1$ if and only if all constraints must be tight in all five matroids, and hence if and only if S' satisfies all constraints of the Slant instance.

Although three-way (and hence also four and five-way) matroid intersection is NP-hard, this formulation implies that any pair of these constraints is solvable in polynomial time, by the (two-way) matroid intersection theorem [29]. For example, if we drop the cycle constraint, and only allow specified numbers on vertices with even coordinate-sums (a checkerboard pattern), then this relaxation of Slant becomes solvable in polynomial time. This particular case also reduces to bipartite b-matching, and even non-bipartite b-matching is well-known to be solvable in polynomial time without matroid intersection methods [25, Theorem 3.5.1]. The result can be more concisely summarized by the following pair of theorems:

Theorem 5 *Let $a, b \in \{0,1\}$ be constants. Suppose we are given a Slant board such that each of its clue with coordinates (x,y) has $x \equiv a \pmod 2$ and $y \equiv b \pmod 2$. Then it can be checked in polynomial time whether the board permits a solution satisfying both the cycle and vertex constraints.*

Theorem 6 *Let $a_1, a_2, b_1, b_2 \in \{0,1\}$ be constants. Suppose we are given a Slant board such that each of its clue with coordinates (x,y) has either both $x \equiv a_1 \pmod 2$ and $y \equiv b_1 \pmod 2$, or both $x \equiv a_2 \pmod 2$ and $y \equiv b_2 \pmod 2$. Then it can be checked in polynomial time whether the board permits a solution ignoring the cycle constraint and satisfying the vertex constraints.*

4 NP-completeness

We will now prove that deciding if a Slant puzzle on an $n \times n$ board is NP-complete. We use a two-step reduction: We first review a (slightly modified) reduction from finding a Hamiltonian cycle with a single required edge in planar, bipartite, 3-regular graph to finding a Hamiltonian cycle with a single required edge in a grid graph [16, 28] (this citation uses a slightly different Hamiltonian cycle variant, so we will add one additional gadget, and show that it is equivalent to their reduction). Then, instead of using the latter problem for another black-box reduction, we will show that we can replicate the functionality of each gadget used, to form a two-step reduction. In this section we will always be using the rotated view of Slant puzzles.

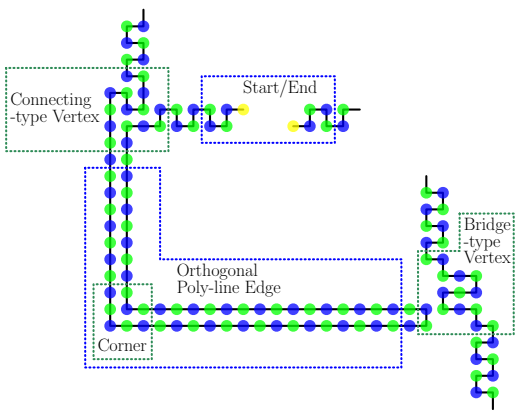


Figure 5: All of the gadgets used in the NP-Hardness construction for Hamiltonian path in grid graphs, and an example partial path going through them.

First, we will briefly summarize the key components of the grid graph reduction; readers interested in a full proof of this reduction's correctness should see [16]. We then make two small modifications to the reduction. Next, we will give gadgets in Slant that force a portion of the graph to be a Hamiltonian cycle along a portion of the grid. Finally, we will address global connectivity issues and complete the proof.

Grid graph reduction overview In Itai, Papadimitriou, and Szwarcfiter's reduction[16] of Hamiltonian cycle/path in grid graphs from Hamiltonian cycle in planar, bipartite, max-degree-3 graphs they take a grid embedding of the planar graph and construct edge and vertex gadgets. These are the vertices in Fig. 5 (note that the grid graph includes all unit-length edges, not just the example path going through the figure). Call the planar graph P and the grid graph G.

Edge gadgets are channels of vertices which are two vertices wide in G and follow the layout of the embedding of P. In a potential Hamiltonian cycle, they can be covered with either a zig-zag path (alternating pairs of left and right turns) or a U-shaped path. A zig-zag path through G has a connection at both ends of the edge gadget and corresponds to that edge gadget being in the Hamiltonian cycle in P. A U-shaped path through G has both ends at the same side of the edge gadget and corresponds to an edge which is not in the

Hamiltonian cycle (however, we still need to cover all of the vertices in G). Vertex gadgets are 3×3 squares of vertices in G and come in two types based on what parity class they belong to in P. Vertex gadgets are split into two classes depending on the part of P from which they were generated. The difference between the classes lies in how they connect to their incident edge gadgets. Fig. 5 provides a visualization of the potential paths and how they interact with the vertex gadgets.

Now we make two small modifications to this reduction. First, we want our graph to be 3-regular, not just max-degree-3. Since this is a strict subproblem and already known to be NP-complete [5], no additional work is needed. Next, we want the problem we are reducing from to have a specified edge which is required to be in the Hamiltonian cycle. This version of the problem is also known to be NP-complete [28], but this is not a strict subproblem. In our proof, we must also require a single edge gadget. Taking inspiration from Itai et al.'s [16] use of degree-1 nodes to force the start and end of a Hamiltonian path, we modify the required edge with the Start/End gadget shown in Fig. 5. The degree-1 vertices in the Start/End gadget will force any Hamiltonian path in G to start and end at those locations and the rest of the edge gadget to be filled with a zig-zag pattern (Fig. 5). With this property, the correctness of the reduction to Hamiltonian path in grid graphs using this gadget follows from the correctness of the Hamiltonian cycle reduction described by Itai et al. We use this gadget instead of the approach of Itai et al. because it will make the second phase of our two-step reduction more straightforward to describe and argue.

Slant reduction With the above construction in mind we now turn to our second phase: how the grid graph reduction can be simulated in Slant. That is, we will now construct a Slant puzzle. Call the grid graph representing the even parity of Slant vertices S. We will at all times assume the Slant puzzle is sufficiently large to admit our construction, which will only require polynomial size. We will first try to ensure that a portion S_G of S must admit a Hamiltonian path to properly satisfy that subset of the puzzle, and later we will discuss connecting the path to the rest of the puzzle to satisfy the cycle constraint of Slant. Most importantly, we will construct portions of S which act as barriers to S_G; it will not be possible to connect them to specific adjacent vertices, thus carving out a subset S_G of S that corresponds directly to G. We will implement the grid vertices in G with degree-2 constrained vertices in S_G, which cannot connect to any vertex in $S \setminus S_G$, and some degree-3 constrained vertices that have exactly one forced connection to a grid point in $S \setminus S_G$ (this will be important to global connectivity, which we will discuss later), and do not permit any others. Additionally, there are exactly two degree-1 grid vertices in G, which will correspond to degree-1 constrained vertices in S_G that act as endpoints to any possible Hamiltonian path. Any valid assignment of edges in S_G satisfying these degree and the cycle constraint will be a Hamiltonian path on S_G, which itself will imply a Hamiltonian path through G. Finally we will address the global connectivity issues in the puzzle that will allow us to satisfy all the constraints of the Slant puzzle if G admits a Hamiltonian path and enforce that any assignment of S that satisfies all the constraints of the Slant puzzle must have a Hamiltonian path over the vertices in S_G.

In order to reliably obtain the behaviour we want from S_G, while admitting a Slant solution if G contains a Hamiltonian path, we will require Slant gadgets that admit each of the gadgets used by Itai et al.. All of the gadgets we will use in this reduction can be seen in Fig. 6. There is a direct correspondence between the gadgets we use for Slant, and the gadgets used in the initial grid graph reduction. That is, we will constrain each part of S_G corresponding to a gadget in G so that its behaviour is the same in any valid solution to the Slant puzzle. The main idea of the construction is to add degree-constrained vertices to S that add *forced* edges. That is, in any valid solution to the final Slant instance we will construct, these edges will be present. Forced edges can be thought of as being possible to "locally deduce", i.e. determine that they are required in any valid solution using only a small neighbourhood around themselves. Most of these forced edges rely on two useful degree-constraint properties to force edges to be in the primal or the dual. One, degree-4 vertices must have all of their adjacent edges. Two, if a degree constrained vertex already has as many adjacent forced primal edges as its constraint allows, then all other edges are forced to be in the dual (and vice versa). The primary backbone of the edge gadgets is a sequence of degree-4 constrained vertices with adjacent degree-1 constrained vertices which are immediately saturated, preventing further connectivity. In Fig. 6, all of the presented gadgets have their corresponding forced edges drawn. Some of these edges require slightly more complex, though straightforward, local deductions:

- Degree-1 constrained vertices in the dual force three surrounding edges to be in the primal.
- If two degree-1 constrained vertices have an edge between them they cannot have any other edges and will thus be disconnected from the rest of the graph (violating the cycle constraint in the dual).
- If we have a pair of degree-1 constrained vertices next to a degree-1 constrained vertex in the dual then all three edges around that vertex in the dual are forced.
- An edge is impossible if one of the endpoint vertices is already saturated.
- A vertex diagonally across from a degree-4 con-

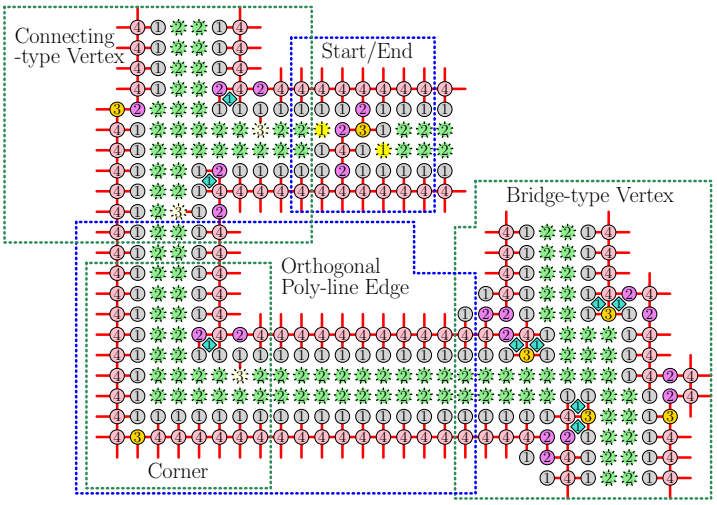


Figure 6: Most of the gadgets used in the NP-Hardness construction for Slant. Circles are vertices with their degree constraint written inside. Green, dotted circles are vertices in S_G. Diamonds are constrained vertices in the dual. Red lines are forced edges.

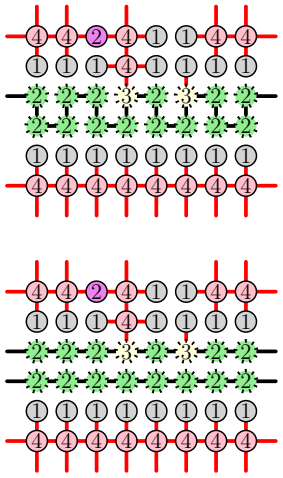


Figure 7: Face connection gadget in the NP-Hardness construction. Forced edges are in red. Two different configurations are given.

strained vertex cannot have two incident edges along the square shared with 4, since doing so would guarantee the formation of a square (a cycle). Hence, if one of these edges is already present, the other is impossible.

- For a given vertex v with edge constraint k, if all but k of its potential edges are impossible, then those edges are forced.

The above set of deduction rules is enough to obtain the entire set of edges of the backbone (i.e., the edges in Fig. 6). In fact, after adding all of the edges forced by degree-4 constrained vertices, each of the remaining backbone edges at a vertex v can be deduced from the constraints contained in the 3×3 window of vertices centred at v using the above deductions. Hence, each deduction step can also be algorithmically implemented with a constant-time brute force check looking at a small window of constraints on a partially-filled board.

Global connectivity. We have three classes of (primal) vertices in S: The vertices in S_G, the degree-specified vertices in $S \setminus S_G$ (i.e., the barrier), and the unspecified vertices in $S \setminus S_G$ (i.e., the face interiors). As specified, the degree-specified vertices in $S \setminus S_G$ will form a cycle around the boundary of each face in P for any possible assignment of edges, which would violate the cycle constraint in the primal. Additionally, there are no connections between the first or second sets of vertices, which would violate the cycle constraint in the dual. We will solve both of these with a gadget simultaneously. To fix this we will assign each face an edge and apply the edge-connection gadget, shown in Figure 7, to connect to that face. If the edge is not long enough to insert the gadget, we can simply scale up the embedding by an at-most polynomial factor. In the connection gadget, the edge above the newly connected degree-3 constrained vertex is forced. Further this is the only place the face is able to connect to the rest of the construction. Thus the number of free edges at that vertex remains 2, the connectivity to the rest of the path is not changed, and thus this does not impact the admissible solutions in the edge. Moreover, this gadget adds a single "gap" in the barrier of each face, removing the cycle. Finally, for the remaining vertices in the puzzle (the unspecified vertices in $S \setminus S_G$), we leave them unconstrained which will allow the space outside our construction to be filled in using Theorem 1 for any valid partial solution to the vertices in S_G and degree-specified vertices in $S \setminus S_G$. There is no possible interaction between the vertices in S_G and the unspecified vertices in $S \setminus S_G$, and hence this does not affect the ability of the vertices in S_G to form a Hamiltonian path (corresponding to a Hamiltonian cycle with a specified forced edge in P).

Now, if the original 3-regular, bipartite, planar Hamiltonian cycle problem has a solution, then so does the Hamiltonian path problem in G. In our Slant instance, we can fill in the vertex and edge gadgets based on the solution in G, fill the faces as connected components with Theorem 1, and fill in the forced edges which gives a valid Slant solution (after rotating 45°).

If there is no solution to the Hamiltonian cycle problem in P, then we know there is no Hamiltonian path in the grid graph G. Our construction ensured that the green vertices which are in the same grid pattern at G only have a solution if there is a Hamiltonian path among those vertices. Thus there would be no solution to the Slant puzzle, completing the reduction.

Combining the prior NP-hardness construction with the observation that checking a potential solution to a Slant puzzle can be done in polynomial time gives us our desired result.

Theorem 7 *Deciding if there is a solution to an $n \times n$ Slant puzzle is NP-complete.*

5 Conclusion and Open Questions

In this paper we've connected the recreational logic puzzle Slant to the Hamiltonian path problem and matroid theory. Through these tools we've shown solving Slant puzzles is NP-complete and given algorithms to solve various special cases and simplifications of Slant. In particular, we showed that the constraints of a Slant puzzle break into five classes, and the simplifications that use just any two of them are solvable in polynomial time. This combination of results leaves open the question of exactly how many/which classes of constraints are needed for NP-completeness. Our construction currently uses all five classes of constraints (the cycle constraint, and all four classes of vertex constraints) and all types of vertex constraints except 0. It would be very interesting to know, for example, if the problem is still hard without the cycle constraint, or if we only need the cycle constraint and vertex constraints in the primal graph. We also know that puzzles which only contain 0 and 4 degree constraints are easy to solve, but what about other subsets?

Another category of questions deals with the uniqueness and quantity of solutions. We can always find a way to fill a partially filled Slant puzzle with no remaining degree constraints, but is there a polynomial time algorithm to count the number of solutions? Further, since having a unique solution is a common design goal in pencil-and-paper puzzles, it is natural to ask: is Slant ASP-hard [35]? Also, is counting the number of solutions #P-hard[33]?

Finally, our view of Slant as a partition of edges between a planar and dual graph naturally leads to a generalization of the puzzle to other types of graphs. Is the problem still NP-hard on the triangular/hexagonal grid? In this case we consider the problem of partition edges between a primal and dual graph; as the interpretation of diagonals does not carry over to triangular girds. Are there interesting and aesthetically pleasing similar puzzles on different forms of planar graphs?

References

[1] ごきげんななめ. `https://www.nikoli.co.jp/ja/puzzles/gokigen_naname/`. Accessed: 2024-04-02.

[2] Z. Abel, J. Bosboom, E. D. Demaine, L. Hamilton, A. Hesterberg, J. Kopinsky, J. Lynch, and M. Rudoy. Who witnesses The Witness? Finding witnesses in The Witness is hard and sometimes impossible. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 3:1–3:21, La Maddalena, Italy, June 2018.

[3] A. Adler, M. Biro, E. Demaine, M. Rudoy, and C. Schmidt. Computational complexity of numberless Shakashaka. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG 2015)*, Kingston, Canada, August 2015.

[4] A. Adler, J. Bosboom, E. D. Demaine, M. L. Demaine, Q. C. Liu, and J. Lynch. Tatamibari is np-complete. In *10th International Conference on Fun with Algorithms*, 2020.

[5] T. Akiyama, T. Nishizeki, and N. Saito. NP-completeness of the Hamiltonian cycle problem for bipartite graphs. *Journal of Information processing*, 3(2):73–76, 1980.

[6] D. Andersson. HIROIMONO is NP-complete. In *Proceedings of the 4th International Conference on FUN with Algorithms*, volume 4475 of *Lecture Notes in Computer Science*, pages 30–39, 2007.

[7] D. Andersson. Hashiwokakero is NP-complete. *Information Processing Letters*, 109(19):1145–1146, 2009.

[8] E. D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 18–33. Springer, 2001.

[9] E. D. Demaine, J. Lynch, M. Rudoy, and Y. Uno. Yin-Yang puzzles are NP-complete. *arXiv preprint arXiv:2106.15585*, 2021.

[10] E. D. Demaine, Y. Okamoto, R. Uehara, and Y. Uno. Computational complexity and an integer programming model of Shakashaka. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E97-A(6):1213–1219, 2014.

[11] E. Friedman. Corral puzzles are NP-complete. `http://www.stetson.edu/~efriedma/papers/corral/corral.html`, August 2002.

[12] E. Friedman. Pearl puzzles are NP-complete. `http://www.stetson.edu/~efriedma/papers/pearl/pearl.html`, August 2002.

[13] M. Holzer, A. Klein, and M. Kutrib. On the NP-completeness of the NURIKABE pencil puzzle and variants thereof. In *Proceedings of the 3rd International Conference on FUN with Algorithms*, pages 77–89, Isola d'Elba, Italy, May 2004.

[14] M. Holzer and O. Ruepp. The troubles of interior design—a complexity analysis of the game Heyawake. In *Proceedings of the 4th International Conference on FUN with Algorithms*, volume 4475 of *Lecture Notes in Computer Science*, pages 198–212, 2007.

[15] A. Ishibashi, Y. Sato, and S. Iwata. NP-completeness of two pencil puzzles: Yajilin and Country Road. *Utilitas Mathematica*, 88:237–246, 2012.

[16] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[17] C. Iwamoto. Yosenabe is NP-complete. *Journal of Information Processing*, 22(1):40–43, 2014.

[18] C. Iwamoto and T. Ide. Five cells and tilepaint are NP-complete. *IEICE TRANSACTIONS on Information and Systems*, 105(3):508–516, 2022.

[19] C. Iwamoto and T. Ide. Moon-or-Sun, Nagareru, and Nurimeizu are NP-complete. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 105(9):1187–1194, 2022.

[20] C. Iwamoto and T. Ide. Chained Block is NP-Complete. *IEICE Transactions on Information and Systems*, 107(3):320–324, 2024.

[21] S. Kanehiro and Y. Takenaga. Satogaeri, Hebi, and Suraromu Are NP-Complete. In *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, pages 46–51. IEEE, 2015.

[22] G. Kendall, A. Parkes, and K. Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

[23] J. Kölker. Kurodoko is NP-complete. *Journal of Information Processing*, 20(3):694–706, 2012.

[24] K. Kotsuma and Y. Takenaga. NP-completeness and enumeration of Number Link puzzle. *IEICE Technical Report*, 109(465):1–7, Mar. 2010.

[25] A. B. Marsh III. *Matching algorithms.* The Johns Hopkins University, 1979.

[26] B. McPhail. The complexity of puzzles. Undergraduate thesis, Reed College, Portland, Oregon, 2003.

[27] B. McPhail. Metapuzzles: Reducing SAT to your favorite puzzle. CS Theory talk, December 2007.

[28] A. Munaro. On line graphs of subcubic triangle-free graphs. *Discrete Mathematics*, 340(6):1210–1226, 2017.

[29] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity.* Courier Corporation, 1998.

[30] H. Tang. A Framework for Loop and Path Puzzle Satisfiability NP-Hardness Results. *arXiv preprint arXiv:2202.02046*, 2022.

[31] C. Thomassen. The Jordan-Schönflies theorem and the classification of surfaces. *The American Mathematical Monthly*, 99(2):116–130, 1992.

[32] R. UEHARA. Computational complexity of puzzles and related topics. *Interdisciplinary Information Sciences*, 29(2):119–140, 2023.

[33] L. G. Valiant. The complexity of enumeration and reliability problems. *siam Journal on Computing*, 8(3):410–421, 1979.

[34] T. Yato. Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. Master's thesis, University of Tokyo, Tokyo, Japan, January 2003.

[35] T. Yato and T. Seta. Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, E86-A(5):1052–1060, 2003. Also IPSJ SIG Notes 2002-AL-87-2, 2002.

Top-k Colored Orthogonal Range Search

Guangya Cai* Ravi Janardan*

Abstract

In a *top-k colored orthogonal range search* problem, a set of n colored, weighted points in $\mathbb{R}^d$ is to be pre-processed into a data structure, so that given an axes-aligned query range q and an integer $k \geq 1$, the k (or fewer) largest-weight points of each color contained in q can be reported efficiently. Efficient data structures are presented in $\mathbb{R}^1$ and $\mathbb{R}^2$ that use near-linear space (up to a low-degree poly-log factor) and answer queries in $O(\log n + \lambda k)$ query time, where λ is the number of distinct colors among the points in q (so λk is an upper bound on the output size). Unfortunately, these results are not practical for large datasets because the poly-log factor in the space bound rapidly overwhelms the available storage. Therefore, practical and space-efficient solutions are also presented. Evaluation of these on large real-world and synthetic datasets show that they achieve good speed-up over baseline data structures (sometimes by up to two orders of magnitude).

1 Introduction

In the *colored orthogonal range search* problem, we are given a set, S, of n points in $\mathbb{R}^d$, where each point has one of $c > 0$ possible colors, representing its "category". We wish to pre-process S into a data structure so that for any axes-aligned query range, q (e.g., an interval in $\mathbb{R}^1$ or a rectangle in $\mathbb{R}^2$), the distinct colors intersected by q can be reported efficiently. (A *color is intersected by* q iff at least one point of that color is intersected by q.) This problem was first proposed in [12] and has been studied extensively. (See the survey [10].)

A natural variant is *top-k colored orthogonal range search*, where each point p in S has a real-valued weight $w(p)$. A query consists of an axes-aligned range, q, and an integer $k \in [1, n]$. We wish to return for each intersected color, the k largest-weight points of that color intersected by q. (If fewer than k points of a color are intersected, then all of them are reported.) The case $k = 1$, i.e., *top-1 colored orthogonal range search*, is an important special case.

This problem has applications in querying databases (e.g., searching a real-estate databases where houses are listed by cost and category—single-family, townhome, condo, etc.—and users wish to identify (say) the 5 most expensive houses in each category that lie in a rectangular query region). Top-k colored search can also provide users with more diverse query results by leveraging the color information (e.g., instead of searching for (say) the 100 most cited papers in CS during some time period, which might favor certain popular areas due to high citation counts, one could ask for (say) the 5 most cited papers in different areas (i.e., categories) of CS).

An easy solution to the problem combines a known solution for orthogonal range search on colored (non-weighted) points [4, 10, 11, 17] with one for top-k orthogonal range search on weighted (non-colored) points [3, 20, 21, 22]. We build a data structure, $\mathcal{T}$, for the former on S and a data structure, $\mathcal{CT}_j$, for the latter on the set of m_j points of color j in S. We query $\mathcal{T}$ to identify the λ distinct colors intersected by q and for each intersected color, j, query $\mathcal{CT}_j$ to report the k (or fewer) points with largest weights. Suppose that $\mathcal{T}$ uses $O(n \log^{O(1)} n)$ space and has query time $O(\log^{O(1)} n + \lambda \log^{O(1)} n)$ and $\mathcal{CT}_j$ uses $O(m_j \log^{O(1)} m_j)$ space and has query time $O(\log^{O(1)} m_j + k)$. Then the combined solution uses $O(n \log^{O(1)} n)$ space with a query time of $O(\log^{O(1)} n + \lambda \log^{O(1)} n + \lambda k)$. (Note that as λk is the worst case output size, that term is unavoidable in the query time.) As a specific example, in $\mathbb{R}^1$ using the real-RAM model, one can use the data structure for colored search in [11] and the data structure for top-k search in [22] to get a solution using $O(n)$ space with a query time of $O(\log n + \lambda \log n + \lambda k)$. In higher dimensions and/or other computational models, one is free to choose the best available data structures for colored search [4, 10, 17] and for top-k search [3, 20, 21, 22] to build a solution.

There are two major issues with this approach. First, it is inefficient for small k due to the $O(\lambda \log^{O(1)} n)$ overhead in the query time. Typically, k is small since users often want to examine only a small subset of "important" records (e.g., if viewing the query output on a small phone screen). Second, the approach may not work well in practice due to the non-linear space usage. For example, even a low-degree poly-log term of $\log^2 n$ in a space bound like $O(n \log^2 n)$ results in a multiplicative factor of ≈ 400 when $n = 10^6$. In practice, linear-size data structures like kd-trees and R-trees are preferred.

We address the first issue by giving (theoretical) solutions in $\mathbb{R}^1$ and $\mathbb{R}^2$ that are very efficient in query

*Department of Computer Science & Engineering, University of Minnesota, Twin Cities, {cai00171,janardan}@umn.edu

time. The query time is $O(\log n + \lambda)$, when $k = 1$ and $O(\log n + \lambda k)$, when $k > 1$. In $\mathbb{R}^1$, the space used is $O(n)$, when $k = 1$, and $O(n \log\log n)$, when $k > 1$. In $\mathbb{R}^2$, the space used is $O(n \log^2 n)$, when $k = 1$, and $O(n \log^3 n)$, when $k > 1$. These results are for the real-RAM model. To address the second issue, we present practical, space-efficient solutions that incorporate kd-trees and R-trees along with certain other tweaks. Experiments on real-world and synthetic datasets show that our solutions achieve good speed-up over baseline data structures that we have implemented.

Related work: Variants of our problem appear in previous work. In [24], the problem of reporting the intersected colors with the k largest weights was considered. (Other approaches to query aggregation were also studied.) Some of these results can be adapted to solve our top-1 colored range search problem. In [18], a problem called Categorical Range Maxima Query (CRMQ) was studied in word-RAM and external memory models. The top-1 colored range search problem was considered over an array of integers using an interval specified by array indices and efficient linear-space solutions were given. To the best of our knowledge, there are no known empirical evaluations of these solutions.

2 Theoretical solutions

Let S be a set of n colored points in $\mathbb{R}^1$ or $\mathbb{R}^2$, where $p \in S$ has a real-valued weight $w(p)$. A query consists of an interval in $\mathbb{R}^1$ or a rectangle in $\mathbb{R}^2$, and an integer $k \geq 1$. For simplicity, we assume that no two points of the same color in S have the same x- or y-coordinate or the same weight. (Otherwise we can use a tie-breaking mechanism based on composite numbers [7].)

2.1 Top-1 search in $\mathbb{R}^1$

We sort the points of each color j in S by increasing x-coordinates into a set S_j and insert dummy points $-\infty$ and $+\infty$ of color j into S_j, both of weight $+\infty$. For each non-dummy point $p_i \in S_j$, we find a pair of points s_i and t_i in S_j such that s_i (resp. t_i) is the closest point to the left (resp. right) of p_i such that $w(s_i) > w(p_i)$ (resp. $w(t_i) > w(p_i)$). We call the open interval (s_i, t_i) the *maximum top-1 range* of p_i.

We wish to obtain a necessary and sufficient condition for p_i to be a top-1 point of color j in q. At a minimum, we must have $p_i \in q$. Additionally, if $q \subseteq (s_i, t_i)$, then any other point of S_j in q has lower weight than $w(p_i)$, so p_i is the top-1 point of color j in q. Conversely, if $q \not\subseteq (s_i, t_i)$, then at least one of s_i and t_i is in q, so p_i is not the top-1 point of color j in q. Thus, p_i is a top-1 point of color j in $q = [a, b]$ iff $p_i \in q$ and $q \subseteq (s_i, t_i)$, i.e., $a \leq p_i \leq b$, $s_i < a$, and $b < t_i$. (This observation is reminiscent of the two-sided chaining method [18] and of an approach to compute a certain type of skyline in a query range [19].)

We transform our problem into a standard (i.e., non-colored) rectangle-stabbing problem, as follows. First, we restate the above conditions as $s_i < a \leq p_i$ and $p_i \leq b < t_i$. Next, we map $q = [a, b]$ to the point (a, b) and map p_i to the rectangle $\mathcal{R}_i = \{(x, y) \in \mathbb{R}^2 \mid s_i < x \leq p_i \text{ and } p_i \leq y < t_i\}$ of color j, both in $\mathbb{R}^2$.

Clearly, p_i is the top-1 point of color j in q iff point (a, b) stabs $\mathcal{R}_i$. The rectangles of color j must be pairwise disjoint (else the top-1 point will not be unique, violating the assumption of distinct weights). Thus, (a, b) stabs at most one rectangle of color j, and that rectangle corresponds to a top-1 point of color j in q. Then, using an efficient rectangle-stabbing data structure [2, 5] built on these rectangles, a top-1 colored search query in $\mathbb{R}^1$ can be answered in $O(\log n + \lambda)$ time and $O(n)$ space.

2.2 Top-k search in $\mathbb{R}^1$ ($k > 1$)

As described in Section 1, we can solve this problem using known data structures for interval range search on colored (non-weighted) points and for top-k interval range search on weighted (non-colored) points. The former uses $O(n)$ space and has a query time of $O(\log n + \lambda)$ [11] and the latter uses $O(n)$ space and has a query time of $O(\log n + k)$ [22]. This yields a solution using $O(n)$ space with a query time of $O(\log n + \lambda \log n + \lambda k)$.

When $k = \omega(\log n)$ the query time of this solution is $O(\log n + \lambda k)$, which is efficient. But when $k = O(\log n)$, the query time is $O(\log n + \lambda \log n) = \Omega(\log n + \lambda k)$. We design an $O(n \log\log n)$-space data structure to answer queries in $O(\log n + \lambda k)$ time when $k = O(\log n)$. These bounds continue to hold for all k when this solution is combined with the solution above for $k = \omega(\log n)$.

Let $\hat{k} = \alpha \log n$, where α is some positive constant. We first design a solution for $k = \hat{k}$ and then show how this can be used to answer queries for any $k \in [2, \hat{k}]$. (Recall that k is part of the query.)

A natural approach is to extend the notion of a maximum top-1 range for a point p_i from Section 2.1 to a maximum top-k range for p_i. However, the description of an exact such range is complicated as there are many possibilities for the relative positions of p_i and the remaining $k-1$ (or fewer) larger-weight points in a query interval. Instead, we resort to defining an approximate top-k range which contains a small superset of the desired top-k points. This superset is returned by the query and we extract the top-k points efficiently from it in a post-processing step, via selection.

As before, let S_j be the points of color j sorted by increasing x-coordinates. For each point $p_i \in S_j$, we scan all points in S_j to the left (resp. right) of p_i and find the k-th point s_i (resp. t_i) with $w(s_i) > w(p_i)$ (resp. $w(t_i) > w(p_i)$). If no such s_i (resp. t_i) exists, we set $s_i = -\infty$ (resp. $t_i = +\infty$). We call the open

interval (s_i, t_i) the *approximate top-k range of* p_i. It can be shown that (see Appendix A.1 for a proof):

Lemma 1 *Let $p_i \in S_j$ and let (s_i, t_i) be its approximate top-k range. Let $q = [a, b]$ be a query interval.*

(i) If $p_i \in q$ and $q \subseteq (s_i, t_i)$, then p_i is among the $2k-1$ (or fewer) largest-weight points of color j in q.

(ii) If $p_i \notin q$ or $q \not\subseteq (s_i, t_i)$, then p_i is not among the k largest-weight points of color j in q.

Lemma 1 does not quite provide a necessary and sufficient condition for p_i to be among the top-k points of color j in q. However, it gives us enough to be useful. Specifically, if the conditions on p_i and q in case (i) are met, then suitable post-processing of the query output (discussed below) allows us to decide if p_i should be reported. If the condition on p_i or on q in case (i) is not met, then case (ii) tells us that we can safely ignore p_i.

As in Section 2.1, we map q to a point (a, b) and each p_i to a rectangle $\mathcal{R}_i$ in $\mathbb{R}^2$ and build an efficient rectangle-stabbing data structure [2, 5] on them. A crucial observation here is that if q intersects a color, then point (a, b) stabs at most $2k-1$ rectangles of that color, by case (i) of Lemma 1. So, all stabbed rectangles can be reported in $O(\log n + \lambda k)$ time.

A final post-processing step is to take the $2k - 1$ (or fewer) points associated with the stabbed rectangles, select the point with the k-th largest weight, and report only those points with weight more than the k-th largest weight. This does not affect the overall query time.

This solution is for a fixed $k = \hat{k} = \alpha \log n$. To handle any $k = O(\log n)$ we use a standard doubling trick. We build the above structure for each value of $k \in [2, \hat{k}]$ that is a power-of-2. This results in $O(\log \log n)$ structures, for a total space bound of $O(n \log \log n)$. To answer a query for some $k \in [2, \hat{k}]$, we query the data structure corresponding to the nearest higher power-of-2, say $\bar{k}$. Since $\bar{k} \leq 2k$, this yields at most $4k-2$ points per color intersected by q. A select-and-scan step is done on these points. The query time remains $O(\log n + \lambda k)$.

Theorem 2 *The top-k colored orthogonal range search problem in $\mathbb{R}^1$ can be solved in $O(n)$ space and $O(\log n + \lambda)$ query time when $k = 1$, and in $O(n \log \log n)$ space and $O(\log n + \lambda k)$ query time when $k > 1$.*

2.3 Top-k search in $\mathbb{R}^2$ ($k \geq 1$)

Extending the previous approach from $\mathbb{R}^1$ to $\mathbb{R}^2$ appears difficult, even for $k = 1$, as the shape of the maximum top-1 range can be irregular and may not be describable by a constant number of inequalities. We propose a different approach based on decomposing the query into quadrants. We describe our solution for $k > 1$ and then show how to simplify it for $k = 1$.

For a generic point $p = (x_p, y_p) \in \mathbb{R}^2$, define its *north-east quadrant*, NE(p), as $\{(x, y) \mid x \geq x_p \text{ and } y \geq y_p\}$ and its *south-west quadrant*, SW(p), as $\{(x, y) \mid x \leq x_p \text{ and } y \leq y_p\}$. Assume that the query range is NE(q), for a point $q = (a, b)$, along with an integer $k > 1$.

Following the idea described in Section 1, the problem can be solved by combining two known data structures. (See [11] and [21], respectively). The solution uses $O(n)$ space and answers queries in $O(\log n + \lambda \log n + \lambda k)$ time. Again, as noted in Section 2.2, the interesting case is when k is "small", i.e. $k = O(\log n)$. So, we fix $\hat{k} = \alpha \log n$, for some positive constant α, and design a solution for $k = \hat{k}$. Subsequently, we show how this solution can be used to answer queries for any $k \leq \hat{k}$.

Fix a color j and let $S_j \subseteq S$ be the set of points of color j. Wlog, assume that no point of S_j has k or more larger-weight points of S_j in its north-east quadrant; otherwise, we identify and remove them in preprocessing. Thus, every point of S_j is among the set of top-k points of color j for at least one query quadrant.

Let $p_1, p_2, \ldots, p_{|S_j|}$ be the points of S_j listed in decreasing order of their weights. Consider a point $p_i = (x_i, y_i)$ in this listing. Our goal is to define the locus of all points, q, such that p_i is among the top-k points for NE(q); we call this the *optimal top-k query region* for p_i, denoted by $Q_k(p_i)$. Let $d_i < k$ be the number of larger-weight points in NE(p_i). Then $Q_k(p_i)$ is the subset of SW(p_i) that is not covered by the south-west quadrants of $k - d_i$ or more of the (larger-weight) points $p_1, p_2, \ldots, p_{i-1}$, i.e., is covered by the south-west quadrants of fewer than $k - d_i$ of the points $p_1, p_2, \ldots, p_{i-1}$.

To build our data structure, we process the points by decreasing weight. For each point, p_i, we extend the horizontal (resp. vertical) boundary edge of SW(p_i) leftward (resp. downward), allowing it to intersect (i.e., cut through) $k - d_i - 1$ vertical (resp. horizontal) boundary edges of previously-processed south-west quadrants and stopping when it meets (i.e., touches) the $(k - d_i)$-th boundary. (If there are fewer than $k - d_i - 1$ intersections, then the corresponding boundary edge of SW(p_i) extends to infinity.)

This results in a rectilinear subdivision, $\mathcal{Z}$, whose vertices are the points of S_j and the intersection and meeting points of south-west quadrant boundaries. (See Figure 1(a).) The number of intersection and meeting points is $O(|S_j|k)$ as there are at most k such meeting points per point of S_j. So the total size of the subdivision is $O(|S_j|k)$. The cells of $\mathcal{Z}$ of interest are those that are below and/or to the left of the closed (staircase-shaped) outer boundary of $\mathcal{Z}$.

To see the reasoning behind this approach, observe that if a boundary edge of SW(p_i) intersects the boundary edge of p_j, $j < i$, it means that a part of SW(p_i) is covered by SW(p_j). Since at most $k - d_i - 1$ such intersections are allowed to happen, there are at most that

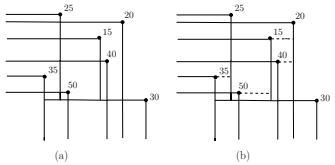


Figure 1: (a) Subdiv. $\mathcal{Z}$; (b) refinement of $\mathcal{Z}$. ($k = 3$.)

many quadrants $\mathrm{SW}(p_j)$, $j < i$, that overlap $\mathrm{SW}(p_i)$. Together with the d_i larger-weight points that are in $\mathrm{NE}(p_i)$, this yields at most $k-d_i-1+d_i = k-1$ larger-weight points for any query quadrant $\mathrm{NE}(q)$ for which q lies in $Q_k(p_i)$. Thus, p_i is among the top-k points of color j for $\mathrm{NE}(q)$.

Call a cell of $\mathcal{Z}$ *rectangle-like* if it is a rectangle, possibly with vertices in the interior of some of its edges. Each cell of $\mathcal{Z}$ is rectangle-like and is the optimal top-k query region, $Q_k(p_i)$, for some point $p_i \in S_j$,

It is possible, however, that some cells of $\mathcal{Z}$ are not rectangle-like. We can convert a non-rectangle-like cell into two or more rectangle-like cells by adding a horizontal line-segment from each corner point of the cell to the nearest vertical boundary to its right. (See Figure 1(b).) The resulting refinement of $\mathcal{Z}$, which we continue to refer to as $\mathcal{Z}$, still has size $O(|S_j|k)$ and all its cells are rectangle-like (either finite or semi-infinite). For each rectangle-like cell of $\mathcal{Z}$, we discard any vertices that lie in edge interiors. This yields a set of $O(|S_j|k)$ interior-disjoint rectangles of color j.

With each of the $O(|S_j|k)$ rectangular cells of $\mathcal{Z}$, we associate a list of the top-k (or fewer) points of color j that are the answer for any query quadrant $\mathrm{NE}(q)$ such that q is in that cell. The list is stored in decreasing order of the weights. We take the set of rectangles so generated for each color in S and pre-process them into a data structure for (standard) rectangle-stabbing queries [2, 5]. The total number of rectangles is $O(nk)$ and each has an associated list of points of size $O(k)$. Therefore, the total space used is $O(nk^2) = O(n\log^2 n)$ since $k = \hat{k} = \alpha \log n$.

This data structure allows us to answer a top-$\hat{k}$ colored query in $O(\log n + \lambda\hat{k})$ time by determining the rectangles stabbed by q in $O(\log n + \lambda)$ time and outputting the points in the associated lists in additional $\lambda\hat{k}$ time.

Recall, however, that we wish to answer queries for any $k \leq \hat{k}$ specified as part of the query. We can do so using the above structure. In the output phase, we simply traverse the sorted lists associated with each stabbed rectangle and output the first k (or fewer) points. Thus, the query time is $O(\log n + \lambda k)$.

It can be shown that the top-k lists at adjacent rectangles differ by at most one point, so the space can reduced to $O(n \log n)$ via (full) persistence without affecting the query time. (See Appendix B.1.) Also, by standard techniques, the solution for quadrant queries can be extended to 4-sided rectangle queries without affecting the query time. (See Appendix B.2.) The resulting data structure uses $O(n\log^3 n)$ space and answers queries in $O(\log n + \lambda k)$. When combined with the earlier discussion for $k = \omega(\log n)$, at the beginning of Section 2.3, we get a solution for all k which uses $O(n\log^3 n)$ space and has a query time of $O(\log n+\lambda k)$.

Deriving the solution for top-1 search in $\mathbb{R}^2$: The solution above for $k > 1$ can be simplified as follows for $k = 1$: (i) There is no need to distinguish between small and large k. (ii) $d_i = 0$, so when building $\mathcal{Z}$ the boundary edges of $\mathrm{SW}(p_i)$ are extended only until they touch the boundary of a previous south-west quadrant, implying that $\mathcal{Z}$ (and its refinement) has size $O(|S_j|)$. (iii) Instead of storing a list of up to k points with each cell of $\mathcal{Z}$, we store a single point that is the answer to $\mathrm{NW}(q)$ if q lies in the cell. So the space for quadrant queries is $O(n)$ and persistence is not needed.

Theorem 3 *The top-k colored orthogonal range search problem in $\mathbb{R}^2$ can be solved in $O(n\log^2 n)$ space and $O(\log n+\lambda)$ query time when $k = 1$, and in $O(n\log^3 n)$ space and $O(\log n + \lambda k)$ query time when $k > 1$.*

We note that there is a subtlety involved in the analysis of the run time in Theorem 3 (and, similarly, in Theorem 2). Due to space limitations, we defer a discussion of this to Appendix B.3.

3 Practical solutions

Even though the data structures above are efficient in query time, they are generally not practical due to the poly-log factor in the space bound (as discussed in Section 1). Here we present practical solutions for top-k colored search in $\mathbb{R}^1$ and $\mathbb{R}^2$.

3.1 Data structure in $\mathbb{R}^1$

In Section 2.1, a linear-size data structure is given for top-1 search which involves rectangle-stabbing. The latter uses a solution from [2, 5], which is not practical. Instead, we solve rectangle-stabbing in a practical way, using an R-tree, augmented with a bulk-loading method called Sort-Tile-Recursive (STR) [15]. Rectangle-stabbing can also be transformed to an orthogonal range query problem in $\mathbb{R}^3$ (see Appendix C.1) and we can use a kd-tree to solve this.

For top-k, $k > 1$, Section 2.2 extended the method for top-1 search and built several rectangle-stabbing-based data structures, alongside an additional data structure for any $k > \hat{k}$, where $\hat{k}$ is dependent on n. In practice, a query optimizer can choose a constant value for $\hat{k}$. Also,

the number of rectangle-stabbing-based data structures can be more flexible. In the most space-constrained situation, one can even build a single data structure from approximate top-$\hat{k}$ ranges to make it work. Of course, the query time becomes worse for small k values. But if memory space allows, one can select a few values between 1 and $\hat{k}$ to build additional data structures to improve the performance.

3.2 Data structure in $\mathbb{R}^2$

Top-k colored search is an extension of orthogonal range search, where a kd-tree [7] is used commonly in practice. Recall that a kd-tree is built by recursively and alternately splitting the given point-set into two subsets of roughly equal size by the median value of a spatial coordinate. In our data structure, since each point has a color, besides splitting by spatial coordinates, splitting into two subsets is also done by using the colors (as explained below). If all points in a subset are of the same color, a substructure for efficient top-k search is built.

Color splitting and substructures: For each color j in S, a minimum axis-aligned bounding box (AABB) is computed for all points of color j. Then the point-set is split into two subsets by splitting AABBs, which we call *color splitting*. There are different criteria to be considered to make a good split, such as overlaps between two subsets of AABBs and tree balancing. In practice, the R-tree and its variants also split a collection of rectangles for an overflowing node. So, we use the method from a variant of the R-tree, namely the R*-tree [1]. Since the two subsets cannot necessarily be split by a single line, an AABB, denoted as *region*(v), is stored at each tree node v to facilitate querying.

We propose two approaches to deploy color splitting: *static splitting* and *greedy splitting*. The first interleaves color splitting with splitting by spatial coordinates. The second considers various criteria and if one criterion is under a threshold value, then color splitting is applied; otherwise, spatial splitting is applied. The criteria include the AABB overlap fraction and the number of colors in the point-set.

While doing spatial or color splitting, if only one point remains then a leaf is created. Moreover, once all points in a subset are of the same color, a substructure for efficient top-k ($k \geq 1$) search is built for those points and the corresponding kd-tree node is denoted as a *single-colored node*. For top-1 search, an augmented kd-tree is built using the approach in [13, 25]. For top-k search ($k > 1$), since it is not obvious how one can adapt this augmentation method, we follow the idea of the counting-based method described in [22]. Here, an augmented kd-tree (storing the sizes of subtrees) is used for range counting queries, and a sorted list of points is maintained to facilitate binary search.

It is not hard to see that our data structure in $\mathbb{R}^2$ is of linear size. It is a combination of linear-size trees, so the number of tree nodes is also $O(n)$. Except for point lists at single-colored nodes, each node uses $O(1)$ space. Finally, each point is stored at most twice (at a leaf and possibly at a single-colored node).

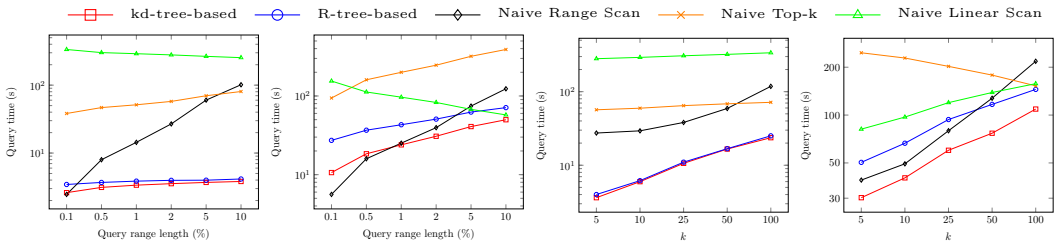
Query algorithm: Let v be a node reached in the query. If *region*(v) only partially intersects the query rectangle q and v is neither a leaf nor a single-colored node, then query proceeds as in a kd-tree (See [7].) The differences lie in the other cases. For top-k search ($k \geq 1$), during the query we maintain a list of top-k candidate points for each color. If v is a leaf and if its point is in q, then the appropriate candidate points list is updated. If v is a single-colored node and *region*(v) is contained in q, the top-k (or fewer) points can be obtained from the list stored at the node. Together with the list of top-k candidate points at hand, top-k points are obtained by merging the two lists and running a linear-time selection algorithm. If q only partially intersects *region*(v), the top-k points can be reported from the substructure and the list of top-k candidate points is updated via the same merge method. If v is neither a leaf nor a single colored node, but *region*(v) is entirely contained in q, we walk down the tree until a leaf or a single colored node is reached and update the list of top-k candidate point accordingly. (See Appendix D.1 for pseudocode and Appendix D.2 for implementation tweaks to get real speed-up in practice.) For top-1 search, the algorithm is simpler as only the top-1 search substructure need be queried at a single-colored node.

4 Experimental results in $\mathbb{R}^1$ and $\mathbb{R}^2$

For better cache performance, all tree-based structures (including those in baseline methods we implemented for comparison) are flattened into linear arrays, with a pre-order traversal layout. (Other layouts might work better [14], but are not explored here.) The query time is measured using the same set of randomly generated 10,000 query ranges for all datasets and is reported as the total time for these query ranges. Unless stated otherwise, we use a query range that occupies 2 percent of the universe (i.e. the maximum span of the points in $\mathbb{R}^1$ or the AABB of the points in $\mathbb{R}^2$) and set $k = 5$. Our implementations were in C++ on an Intel Core i5-8300H (2.30 GHz) machine with 16GB RAM. Due to space constraints, only results on real-world datasets are discussed here. (See Appendices E.4 and F.3 for results on synthetic datasets in $\mathbb{R}^1$ and $\mathbb{R}^2$, respectively.)

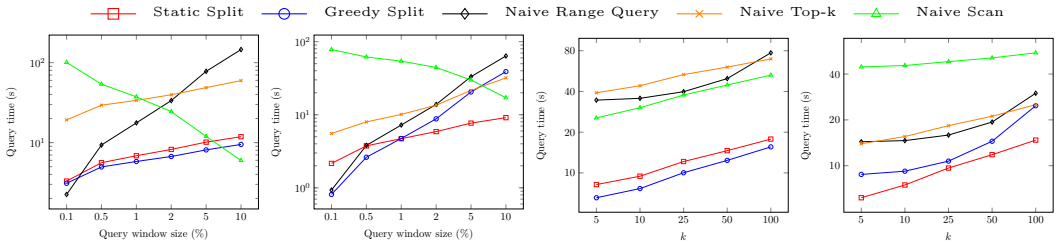
4.1 Results in $\mathbb{R}^1$

The datasets we use are *Google Local Dataset* [16] (n = 38.9M, c = 3455) and *Stack Overflow Questions*



(a) Google Local (b) Stack Overflow Q's (c) Google Local (d) Stack Overflow Q's

Figure 2: Query time in $\mathbb{R}^1$ as a function of range length ((a), (b)) and k ((c), (d)). (Plot best viewed in color.)



(a) CalCROP21 (b) Chicago Crime (c) CalCROP21 (d) Chicago Crime

Figure 3: Query time in $\mathbb{R}^2$ as a function of window size ((a), (b)) and k ((c), (d)). (Plot best viewed in color.)

Dataset [23] (n = 17.7M, c = 35751), where n and c are number of points and colors, respectively. (See Appendix E.1 for details.) For comparison, three baseline methods, namely *Naive Range Scan*, *Naive Top-k* and *Naive Linear Scan* were implemented. (See Appendix E.2.) Throughout, our queries are handled by the rectangle-stabbing-based data structures used in our method. For large k values, one can choose one of the baseline methods. In this case the query time comparison is uninteresting.

In Figures 2(a) and 2(b), our kd-tree-based method performs best for any sufficiently large query range and our R-tree-based method also shows good performance for the Google Local dataset. In the larger dataset (Google Local), the kd-tree-based method achieves speedups of up to 21x, 27x and 128x over Naive Top-k, Naive Range Scan and Naive Linear Scan, respectively. We notice that our method has a greater advantage in the Google Local dataset than in the other one. Our query algorithm consists of two parts: (1) Rectangle-stabbing and (2) Gathering and selecting top-k points per intersected color. We observe that as the query range becomes larger, the time spent in part (2) grows more rapidly than the time in part (1). For the Stack Overflow Questions dataset, the issue is worse since it has more colors.

Figures 2(c) and 2(d) shows the query time as a function of k. To show the peak performance, the data structure built from approximate top-k ranges is queried for each input k. (See Appendix E.3 for results in a different setting.) Our kd-tree-based method performs consistently the best and our R-tree-based method performs better than baseline methods for one dataset. We notice that the query time of our methods increases more rapidly for the Google Local dataset than for the other dataset. This is because the number of colors of the latter dataset is larger, which results in a proportionately slower growth of output size as k increases.

4.2 Results in $\mathbb{R}^2$

The datasets we use are *CalCROP21 Dataset* [9] (n = 19.2M, c = 70) and *Chicago Crime Dataset* [6] (n = 7.5M, c = 32). (See Appendix F.1 for details.) For the static splitting approach, we apply two color splits consecutively after two spatial splits (x and y). For the greedy splitting approach, we take the overlap threshold value to be 0.5 and the color count threshold value to be 4. For comparison, three baseline methods, namely *Naive Range Query*, *Naive top-k* and *Naive Scan* were implemented. (See Appendix F.2.) Query windows with aspect ratio ranging from 1/4 to 4 are used.

Figures 3(a) and 3(b) shows the query time as a function of the query window size. For CalCrop21, both static splitting and greedy splitting work well and outperform all other methods in almost all cases. Greedy splitting performs slightly better than static splitting. Since points of the same color are more clustered in this dataset, greedy splitting is able to make good splits. In this larger dataset, greedy splitting is able to achieve speedups of up to 6x, 15x and 33x over Naive top-k, Naive Range Query and Naive Scan, respectively. For

the Chicago Crime dataset, static splitting outperforms baseline methods except for the smallest query window. Even though the clustering of data of the same type is less obvious here, greedy splitting still performs consistently better than Naive Range Query.

In Figures 3(c) and 3(d), the query times of all methods increase mildly with k. For CalCROP21, both static splitting and greedy splitting perform consistently better than other baseline methods, with the greedy splitting being the best. For the Chicago Crime dataset, the static splitting method performs the best.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[2] A. Boroujerdi and B. M. Moret. Persistency in computational geometry. In *Proc. 7th Canadian Conf. Comp. Geometry (CCCG 95)*, pages 241–246, 1995.

[3] G. S. Brodal, R. Fagerberg, M. Greve, and A. López-Ortiz. Online sorted range reporting. In *Algorithms and Computation: 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings 20*, pages 173–182. Springer, 2009.

[4] T. M. Chan and Y. Nekrich. Better data structures for colored orthogonal range reporting. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 627–636. SIAM, 2020.

[5] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

[6] Chicago Police Department. Reported incidents in the city of Chicago, 2023. Retrieved from UCR-STAR `https://star.cs.ucr.edu/?Chicago%20Crimes`.

[7] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and applications*. Springer, Heidelberg, Germany, 3rd edition, 2008.

[8] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

[9] R. Ghosh, P. Ravirathinam, X. Jia, A. Khandelwal, D. Mulla, and V. Kumar. CalCROP21: A georeferenced multi-spectral dataset of satellite imagery and crop labels. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1625–1632. IEEE, 2021.

[10] P. Gupta, R. Janardan, S. Rahul, and M. Smid. Computational geometry: Generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, pages 1043–1058. Chapman and Hall/CRC, 2018.

[11] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

[12] R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3(01):39–69, 1993.

[13] M. Jurgens and H.-J. Lenz. The R_a^*-tree: an improved R*-tree with materialized data for supporting range queries on OLAP-data. In *Proceedings Ninth International Workshop on Database and Expert Systems Applications*, pages 186–191. IEEE, 1998.

[14] P.-V. Khuong and P. Morin. Array layouts for comparison-based searching. *ACM Journal of Experimental Algorithmics*, 22:1–39, 2017.

[15] S. T. Leutenegger, M. A. Lopez, and J. Edgington. STR: A simple and efficient algorithm for R-tree packing. In *Proceedings 13th International Conference on Data Engineering*, pages 497–506. IEEE, 1997.

[16] J. Li, J. Shang, and J. McAuley. UCTopic: Unsupervised contrastive learning for phrase representations and topic mining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 6159–6169, 2022.

[17] Y. Nekrich and J. S. Vitter. Optimal color range reporting in one dimension. In *European Symposium on Algorithms*, pages 743–754. Springer, 2013.

[18] M. Patil, S. V. Thankachan, R. Shah, Y. Nekrich, and J. S. Vitter. Categorical range maxima queries. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 266–277, 2014.

[19] S. Rahul and R. Janardan. Algorithms for range-skyline queries. In *SIGSPATIAL 2012 International Conference on Advances in Geographic Information Systems (formerly known as GIS), SIGSPATIAL'12, Redondo Beach, CA, USA, November 7-9, 2012*, pages 526–529. ACM, 2012.

[20] S. Rahul and R. Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):2859–2871, 2014.

[21] S. Rahul and Y. Tao. On top-k range reporting in 2d space. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 265–275, 2015.

[22] S. Rahul and Y. Tao. A guide to designing top-k indexes. *ACM SIGMOD Record*, 48(2):6–17, 2019.

[23] D. Robinson. Stacklite: A simple dataset of stack overflow questions and tags. `https://github.com/dgrtwo/StackLite`, 2017.

[24] B. Sanyal, P. Gupta, and S. Majumder. Colored top-k range-aggregate queries. *Information Processing Letters*, 113(19-21):777–784, 2013.

[25] Y. Tao and D. Papadias. Range aggregate processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1555–1570, 2004.

A Omitted details of Section 2.2

A.1 Proof of Lemma 1

Proof. Suppose that $p_i \in q$ and $q \subseteq (s_i, t_i)$. Then there can be at most $k-1$ points on either side of p_i with weight larger than $w(p_i)$. Thus, p_i is among the $2k-1$ (or fewer) largest-weight points of S_j in q. On the other hand, if $p_i \in q$ and $q \not\subseteq (s_i, t_i)$, then at least one of s_i and t_i is in q. Thus there are at least k larger-weight points of color j in q, so p_i is not among the k largest-weight points of color j in q. Of course, if $p_i \notin q$, then regardless of whether or not $q \subseteq (s_i, t_i)$, p_i is not among the k largest-weight points of color j in q. □

B Omitted details of Section 2.3

B.1 Reducing space via persistence

Consider the subdivision $\mathcal{Z}$ for some color j. We build a graph, G, where vertices correspond to cells of $\mathcal{Z}$ and edges join vertices whose cells share an edge. Observe that when we traverse an edge (u, v) of G, we either enter some south-west quadrant or exit it. Thus, the lists of top-k (or fewer) points stored with the cells associated with u and v differ by at most one point. So, we can store all the lists of $\mathcal{Z}$ compactly via full persistence [8].

We do a depth-first search of the graph G derived from $\mathcal{Z}$. When we traverse an edge (u, v), from u to v, we update the list associated with u in a fully persistent way, by inserting or deleting one point, to obtain the list for v. At the end of the search, we have a fully persistent data structure storing all the lists of $\mathcal{Z}$. There are $(|S_j|k)$ updates in total, so by the result in [8], the overall space used by all the lists in $\mathcal{Z}$ is $O(|S_j|k)$. Thus, the total space used for all colors is $O(nk) = O(n \log n)$, since $k = \hat{k} = \alpha \log n$.

B.2 Handling 3- and 4-sided rectangles

Suppose that the query rectangle, $R_{\sqcup}$, is 3-sided and upward-unbounded. We build a binary search tree on S by bisecting it recursively with vertical lines and storing at each node instances of the north-east (and a symmetric north-west) quadrant data structure from Section 2.3. Given $R_{\sqcup}$, we find a node of the tree whose line intersects it, dividing it into a north-east and a north-west quadrant. We query the associated structures to obtain a set of at most $2k$ largest-weight points in $R_{\sqcup}$. We identify the k largest-weight ones among these points by first selecting the k-th largest point in $O(k)$ time. We scan the set of at most $2k$ points and output the ones that have larger weight, as well as the k-th largest point. The query time is $O(\log n + \lambda k)$ and the space used is $O(n \log^2 n)$.

For 4-sided rectangles, we build a binary search tree on S by bisecting it with horizontal lines. Each node stores instances of the above 3-sided query structure, for upward- and downward-unbounded rectangles. A 4-sided query is decomposed into two 3-sided ones and the appropriate associated structures are queried, followed by a select-and-scan step as above.

The resulting two-level tree structure adds two log-factors to the space bound of the quadrant structure, taking it to $O(n \log^3 n)$, but the query time remains $O(\log n + \lambda)$.

B.3 Making query times truly output-sensitive

Let l_j be the number of points of color j in S returned by a top-k colored query with q. For $k > 1$, the query time of $O(\log n + \lambda k)$ achieved by our query algorithms can be expressed more precisely as $O(\log n + \sum_j l_j)$, where the summation is over all intersected colors. This accounts for the possibility that $l_j \ll k$ for some (possiblly all) of the intersected colors j and makes the query time truly output-sensitive.

A careful study of our query algorithms in $\mathbb{R}^1$ and $\mathbb{R}^2$ for small k (i.e., $k \leq \hat{k} = O(\log n)$) shows that they in fact achieve this bound. (Specifically, in $\mathbb{R}^1$, the number of stabbed rectangles of color j is at most $2l_j - 1$ and in $\mathbb{R}^2$ the traversal of the sorted list for the stabbed rectangle of color j visits and outputs l_j points.)

For large k (i.e., $k > \hat{k}$), however, the query algorithm used (by combining known results for colored range search and non-colored top-k search) has a run time which includes the term $\log n + l_j$ for each of the λ intersected colors j. (This term comes from the non-colored top-k query.) When $l_j \ll k$, the $\log n$ term is not subsumed by the l_j term, which makes the query time non-output-sensitive.

Fortunately, this issue can be resolved in a simple way, as follows. When $k > \hat{k}$, we first query the data structure designed for small k values by setting $k = \hat{k}$. Then, in the output of the query, we count the number of points per intersected color. If for some color j the number of output points is $\hat{k}$, then the non-colored top-k search data structure for that color is queried to report all top-k points. In this case, since $\hat{k} = \alpha \log n$, we know that $l_j = \Omega(\log n)$ so the additional time spent on non-colored top-k search for color j is $O(\log n + l_j) = O(l_j)$. For those colors where the number of output points is less than $\hat{k}$, queries on the non-colored top-k search structures are not needed since all the relevant points have already been output. Thus, the total time spent on non-colored top-k search for all intersected colors is $O(\sum_j l_j)$ and the query time becomes $O(\log n + \sum_j l_j)$, as desired.

For simplicity, and due to lack of space, this issue is ignored in the body of the conference submission and the query time is written as $O(\log n + \lambda k)$ throughout.

C Omitted details of Section 3.1

C.1 Further transformation of rectangle-stabbing

Recall our transformation from Section 2.1: For a given query range $q = [a, b]$ and a point p_i with its maximum top-1 range being (s_i, t_i), q is transformed into a point (a, b) and p_i is transformed into a rectangle $\mathcal{R}_i = \{(x, y) \in \mathbb{R}^2 \mid s_i < x \leq p_i \text{ and } p_i \leq y < t_i\}$. The point (a, b) can be further transformed into a hyperrectangle $\mathcal{Q} = \{(x, y, z, w) \in \mathbb{R}^4 \mid x < a, y \geq a, z \leq b, w > b\}$ in $\mathbb{R}^4$ and $\mathcal{R}_i$ can be transformed to a point (s_i, p_i, p_i, t_i) in $\mathbb{R}^4$. Then the original top-1 point from Section 2.1 can be solved by doing an orthogonal range query with $\mathcal{Q}$ on the set of points in $\mathbb{R}^4$ resulting from the transformation. Notice that the point (s_i, p_i, p_i, t_i) in $\mathbb{R}^4$ has identical second and third coordinates, i.e., p_i. This allows us to replace the two separate constraints $a \leq p_i$ and $b \geq p_i$ by a single constraint $a \leq p_i \leq b$. Thus, we can work in $\mathbb{R}^3$ with the set of points (s_i, p_i, t_i) and the query range $\{(x, y, w) \in \mathbb{R}^3 \mid x < a, a \leq y \leq b, w > b\}$.

D Omitted details of Section 3.2

D.1 Pseudocode for the query algorithm

Algorithm 1 Query algorithm for top-k search ($k \geq 1$)

Input: A node v in the data structure, query rectangle q, and collection, L, of lists of top-k candidate points for each color ($L[j]$ is the list for color j).
Output: Top-k points lists stored in L.

```
1: procedure QUERY(v, q, L)
2:     if v is a leaf then
3:         Update the appropriate top-k candidate list
           in L if the point stored in v lies in q.
4:     else if v is a single colored node then
5:         l ← L[color(v)]
6:         if region(v) is fully contained in q then
7:             l' ← top-k weighted points stored at v
                   or all points if the list has less than
                   k points.
8:             MERGEANDSELECTTOPK(l, l')
9:         else if region(v) intersects q then
10:            l' ← top-k weighted points obtained by
                   querying the substructure.
11:            MERGEANDSELECTTOPK(l, l')
12:        end if
13:    else if region(v) is fully contained in q then
14:        UPDATETOPKCANDIDATES(lc(v), L)
15:        UPDATETOPKCANDIDATES(rc(v), L)
16:    else if region(v) intersects q then
17:        QUERY(lc(v), q, L)
18:        QUERY(rc(v), q, L)
19:    end if
20: end procedure
```

D.2 Implementation tweaks

The query algorithm presented in Section 3.2 shows the general idea of the query algorithm. However, to get real speed-up in practice, there are still a number of implementation details one has to be careful about.

For the top-k search substructure, range counting is used to find the threshold weight. However, it is not necessary to count exactly all the time; one only has to determine whether the number of points is larger than, less than, or equal to k while doing binary search. In this case, one can terminate the counting process once it is known that the number of points inside the query box is larger than k. This simple trick improves the performance of the counting-based top-k search method significantly.

Also, alongside the top-k candidate points, we keep track of the minimum weighted one among them, whose weight is denoted w_m^j. This enables fast testing of points stored at leaves, where a point can be rejected efficiently if its weight is less than w_m^j. Also, for top-k search at the single colored node, one can use w_m^j to test against the maximum weighted point stored at this node and skip all subsequent operations if w_m^j is larger. Moreover, one can use it to do range counting first along the third dimension using the range $[w_m^j, +\infty)$ and report all points if the number of points inside the range is less than k. Finally, w_m^j can be used as a starting point for binary search for the threshold weight, or as a stopping point if the AABB of a single colored node is fully contained in q.

One last trick worth mentioning to practitioners (but which was not used in our experiments) is that one can further speed-up our data structure by substituting some counting-based substructures with simple sorted lists of points, when the number of points in the substructure is under some threshold value. Our choice of the substructure used for top-k search introduces a log-factor in the query time, which makes it inefficient when the number of points in the substructure is small. However, the choice of the threshold value could be highly dependent on the machine used and on the data distribution, which makes the problem challenging.

E Omitted details and additional experimental results of Section 4.1

E.1 Datasets

Details on datasets we use in this work for top-k colored searching in $\mathbb{R}^1$ are listed below, where n denotes the number of data points and c denotes the number of colors.

1. *Google Local Dataset* [16] : This dataset contains review information on Google map (ratings, text,

images, etc.), business metadata (address, geographical info, descriptions, category information, price, open hours, and other miscellaneous information) up to Sep 2021 in the United States. The timestamp of a review is regarded as an attribute to be range-queried since one might only be interested in reviews within a certain time period. The category of the business is regarded as a color and the length of the text is regarded as a weight. Note that a business might be in different categories at the same time (e.g. used car dealer and used truck dealer). In our experiments, multiple data points, one for each color, are created from a single review. Since the dataset is large, with 666 million reviews, we only use data for Minnesota and Wisconsin. Reviews without text are filtered out and we end up with $n = 38.9 \times 10^6$ and $c = 3455$.

2. *Stack Overflow Questions Dataset* [23]: This dataset contains all questions asked by users over the years at `stackoverflow.com`. The timestamp for the creation of a question is regarded as an attribute to be range-queried. For each question, up to five tags can be added to it and these tags are regarded as colors in our experiments. Since each question might have multiple colors, multiple data points, one for each color, are created from a single question. Besides, users can up-vote a question if they think it is a good one or down-vote it if they think the other way. A score equal to the total number of up-votes minus the total number of down-votes is assigned to each question, and it is regarded as the weight of the question. In our experiments, we used all questions asked between 2008 and 2013 and filtered out all deleted questions. We ended up with $n = 17.7 \times 10^6$ and $c = 35751$.

3. *Uniform Synthetic Dataset:* All coordinates and weights of the points are generated uniformly at random in the interval $[-500, 500]$ and a color is assigned to each point uniformly at random. We have $n = 20.0 \times 10^6$ and $c = 200$ or $c = 1000$.

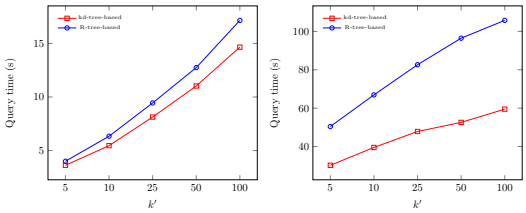
E.2 Baseline methods

For comparison, several baseline data structures were implemented for top-k colored searching in $\mathbb{R}^1$:

1. *Naive Range Scan:* A list of points sorted by their x-coordinate in increasing order is maintained. For a given query range $q = [a, b]$, the first point in the list whose x-coordinate is greater than or equal to a is identified by a binary search, followed by a linear scan to update the list of top-k candidate points for each encountered color until a point whose x-coordinate is greater than b, or the end of list, is reached. Note that we do not report all points and select top-k points since this method performs poorly for large query ranges and small k due to significantly larger number of memory allocations and worse cache performance.[1] Besides, we also keep track of the minimum weighted point among top-k candidate points per intersected color for fast point rejections, as done in Section D.2.

2. *Naive Top-k:* For each color, a counting-based top-k search data structure is built on points of that color, alongside a list of points sorted by their x-coordinate in increasing order. A query first determines whether there are at most $2k$ points inside the query range by two binary searches on the sorted list. If so, the top-k points are determined via selection among up to $2k$ points. Otherwise, the counting-based top-k search data structure is queried to find the threshold weight and report the top-k points. (The early termination trick applied in Section D.2 is also used here.)

3. *Naive Linear Scan:* For each color, a list of points sorted by their weights is maintained. To answer a query, each list is scanned in sorted order until either k points inside the query range are found or the end of the list is reached.



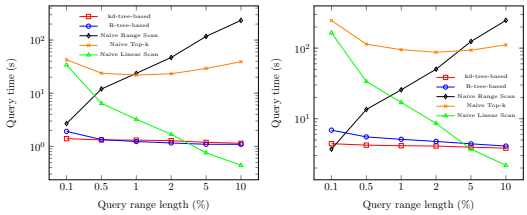
(a) Google Local (b) Stack Overflow

Figure 4: Query time versus approximate top-k' ranges for real-world datasets.

E.3 Effect of approximate top-k' ranges

In our experiments, we use k' to denote an integer in $[1, \hat{k}]$ for which we build an instance of our rectangle-stabbing-based data structures. Figure 4 shows that query time of our data structures that are built with different k' values, but only report top-5 points per intersected color. For a better illustration, the query time (y-axis) is on a linear scale. Also, for a better performance, for any $k' > 5$, instead of gathering and selecting top-k points per intersected color, the implementation is changed to keep updating the top-k candidate list per color while doing rectangle-stabbing, which is similar

[1] We observe that the report-and-select method performs better if the output size is close to the number of points in q. In most of our experiments, the output size is significantly smaller.



(a) Uniform ($c = 200$) (b) Uniform ($c = 1000$)

Figure 5: Query time versus query range length for synthetic datasets.

to the implementation in Naive Range Scan. We observe that while k' grows 20x, the query time of the kd-tree-based method only increases 4.0x and 2.0x for the Google Local and Stack Overflow Questions datasets, respectively, and the query time of the R-tree-based method increases 4.3x and 2.1x, respectively. Moreover, one can also notice that the query time increases more slowly for the Stack Overflow Questions dataset. This is because the number of stabbed rectangles also increases more slowly, which is consistent with what we have observed in the experimental results for varying values of k. This experiment show that with a proper implementation, the query time will not increase dramatically by querying the data structure built from approximate top-k' ranges with a large k' value.

E.4 Experimental results on synthetic datasets

Figure 5 shows the query time of all our data structures as a function of the query range length for uniform datasets. Naive Range Scan performs well when the query range length is very small, but its query time increases drastically as the query range length increases. On the contrary, Naive Linear Scan performs well when the query range length is very long, but its query time increases drastically as the query range length decreases. The performance of our kd-tree-based and R-tree-based methods is more stable across different query range lengths and both outperform Naive Range Scan and Naive Linear Scan in most cases. Also, compared with Naive Top-k, our methods consistently performs better.

As for the comparison between the kd-tree-based and R-tree-based method: When the number of color is large or the query range is small, the kd-tree-based method performs better, which is consistent with observations from experimental results on real-world datasets (Section 4.2).

E.5 Space usage

Table 1 shows the space usage of our data structures, for different datasets. Note that for our data structures, only the space usage for the one built with the largest k' value in our experiments ($k' = 100$) is shown.

	Uniform ($c = 200$)	Google Local	Stack Overflow Questions
kd-tree-based	1525.88	2969.55	1350.63
R-tree-based	1831.05	3563.46	1620.75
Naive Range Scan	457.76	890.98	405.19
Naive Top-k	1831.08	3564.29	1625.40
Naive Linear Scan	305.18	594.09	271.22

Table 1: Comparison of space usage of data structures in $\mathbb{R}^1$ (in MB)

As we have discussed in Section 3.1, the actual space usage can be varied to get a space-time trade-off. In fact, the one we show here is the most space-consuming one among data structures built for different k' values. In the datasets we used, multiple points could be located at the same coordinate and a point could be discarded in the pre-processing if there are more than k' larger weighted points of the same color and the same coordinate. For a smaller k', a point is more likely to be discarded. Comparing our kd-tree-based implementation and R-tree-based implementation, the R-tree uses more space because intermediate tree nodes store AABBs, which are space-consuming. Even though our data structures will not be space-efficient if one hopes to improve the query time by using more space, given the benefits realized in query efficiency we believe that this space overhead is reasonable.

F Omitted details and additional experimental results of Section 4.2

F.1 Datasets

Details on datasets we use in this work for top-k colored searching in $\mathbb{R}^2$ are listed below, where, as before, n denotes the number of data points and c denotes the number of colors.

1. *CalCROP21 Dataset* [9]: The dataset contains a number of images covering the state of California, where each pixel of the image corresponds to a 10m×10m region. At each pixel, there is an (interpolated) Cropland Data Layer (CDL) label representing the type of the crop and 10 satellite spectral signatures over 24 timestamps. A CDL label is regarded as a color and a satellite spectral signature (a real number) at a certain timestamp is regarded as a weight. Since the dataset is large (with 442 million points), we only use a subset of it. We also filter out some invalid and irrelevant (e.g. water and open water) CDL labels and we end up with $n = 19.2 \times 10^6$ and $c = 70$.

2. *Chicago Crime Dataset* [6]: This dataset reflects reported incidents of crime in the City of Chicago

starting from 2001 to Sep 2023. Every crime report contains location, time, crime type information. Each crime type is regarded as a color. We use the time of a reported case as the weight so that one can query k most recent reports in each crime type. Records with incorrect and missing values are filtered out and we end up with $n = 7.5 \times 10^6$ and $c = 32$.

3. *Uniform Synthetic Dataset:* All coordinates and weights of the points are generated uniformly at random in the interval $[-500, 500]$ and colors are also assigned to the points uniformly at random. We have $n = 10.0 \times 10^6$ and $c = 50$ or $c = 200$.

4. *Clustering synthetic dataset:* In this dataset, points of the same color are generated around a randomly generated center using the Gaussian distribution. Coordinates of cluster centers and weights of points are generated uniformly at random in the interval $[-500, 500]$. The standard deviation of the Gaussian distribution is set to be 20 and the minimum distance between two centers is also 20. We have $c = 50$, which is also the number of colors. As each cluster consists of $n = 2.0 \times 10^5$ points, we have $n = 10.0 \times 10^6$.

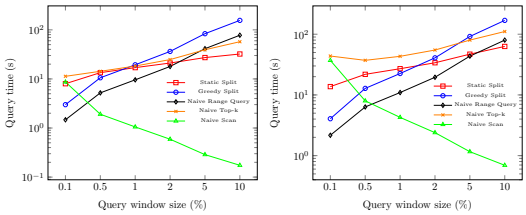
F.2 Baseline methods

For comparison, several baseline data structures were implemented for top-k colored searching in $\mathbb{R}^2$:

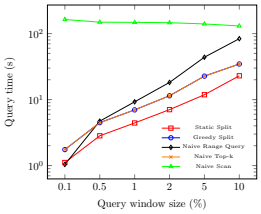
1. *Naive Range Query:* In this method, a kd-tree is built on the point-set and we do a standard range query with the given query window q. During a query, the list of top-k candidate points for each color is updated while visiting the leaves. Note that we do not report all points and select top-k points, for the same reason in the similar method for colored top-k search in $\mathbb{R}^1$. Besides, minimum weighted point among top-k candidate points per intersected color is also maintained for a further speed-up.

2. *Naive top-k:* For each color, a counting-based top-k search data structure is built on points of that color. Top-k colored points are reported by querying each top-k search data structure (with the same early termination trick from Section D.2 applied).

3. *Naive Scan:* For each color, a list of points sorted by their weights is maintained. During a query, each list is scanned from the beginning till either k points inside q are found or the end of the list is reached.



(a) Uniform ($c = 50$) (b) Uniform ($c = 200$)



(c) Clustering ($c = 50$)

Figure 6: Query time versus query window size for synthetic datasets.

F.3 Experimental results on synthetic datasets

We have synthetic datasets with two data distributions in our experiments, where one is uniformly distributed and one is more clustered. For the uniform dataset, any sufficiently large subset of the dataset will contain all colors and the AABBs for points of the same color will have large overlaps with each other. In this case, our static color splitting method will not result in good splits and our greedy color splitting method will not perform any color splitting until the subset of points is small enough. However, our static splitting method still shows some advantages over the baseline method for some query window sizes.

Figure 6 shows the query time of all data structures as a function of the query window size. For uniform datasets, our data structure with the greedy splitting method performs consistently worse than Naive Range Query. This is because color splitting can only be done for very deep nodes where only a very small number of points are stored in the top-k search substructure; this actually worsens the performance of top-k search due to the log factor in query time introduced by the threshold weight-finding step. However, our data structure with the static splitting method outperforms Naive Range Query for sufficiently large query windows. Even though static splitting gives us two regions with large overlaps, a larger query window is more likely to overlap regions of both children.

For the clustering dataset, the performance of our two data structures is much better, when compared to the results for the uniform dataset. Since the dataset is more clustered for each color, our greedy split and the static split methods are more likely to make good color splits, which results in the speed-up. Note that

	Uniform ($c = 50$)	Clustering	CalCROP21	Chicago Crime
Static Split	844.21	839.28	1612.64	636.54
Greedy Split	1081.86	839.24	1612.59	704.40
Naive Range Query	610.35	610.35	1170.06	461.96
Naive top-k	839.24	839.24	1608.84	635.20
Naive Scan	228.88	228.88	438.77	173.24

Table 2: Comparison of space usage of data structures in $\mathbb{R}^2$ (in MB)

the Naive Top-k method is quite close to our greedy split method. The reason is that since points of the same color are more clustered, spatial splittings are seldom performed in our greedy split method. The resulting data structure can be viewed as an R-tree built on the top of multiple top-k search data structures. Since $c = 50$ is not a large number and clusters are separated by a minimum distance, the performance is similar with or without an R-tree for pruning.

Even though for the synthetic datasets our data structures only show performance gains for large query windows or clustered data sets, and the greedy splitting method does not perform too well, our experiments on real-world datasets show more promising results (as seen in Section 4.2).

F.4 Space usage

Table 2 shows the space usage of all data structures for different datasets. Notice that the static splitting method and the greedy splitting method have similar space usage in two datasets but have different space usage in the other two, which might seems odd considering both data structures scale linearly in terms of the number of points n. However, recall that our structure is a combination of two linear-size data structures, and tree nodes of the main structure have to store AABBs for color splitting while substructures are implemented as kd-trees, where only a single splitting line is needed at each intermediate tree node. This means that a node of the main structure is more space-consuming than a node of the substructure. For the two splitting methods, differences in space usage actually reflects differences in the substructures built. Also, given the benefit realized in query efficiency, we believe that the space overhead here is reasonable.

Set Cover and Hitting Set Problems for Some Restricted Classes of Rectangles *

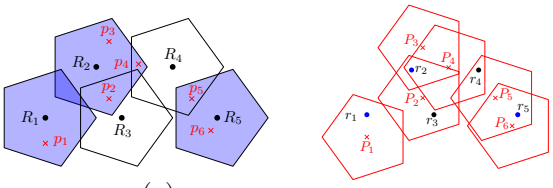
Minati De† Ratnadip Mandal‡ Subhas C. Nandy§

Abstract

We consider the well-known (weighted/unweighted) set cover and hitting set problems for some restricted classes of rectangles. We show that the (unweighted) set cover problem for boundary rectangles is APX-hard and also give an 8-approximation algorithm for the problem. For the (weighted) set cover problem of cross-separable rectangles, we provide a 2-approximation algorithm. Finally, we prove that both the (weighted) set cover and hitting set problems of translated copies of a convex object touching a diagonal line from the right side can be solved in polynomial time.

1 Introduction

In computational geometry, set cover and hitting set problems are amongst the most important problems due to their wide range of applications in wireless networks, VLSI design, resource allocation, image processing, sensor networks, database systems, computer vision etc [1, 14]. A *range space* $\Sigma = (\mathcal{X}, \mathcal{R})$ consists of a ground set $\mathcal{X}$ of elements and a family $\mathcal{R}$ of subsets of $\mathcal{X}$. A *set cover* of a given range space $\Sigma = (\mathcal{X}, \mathcal{R})$ is a subset $\mathcal{S} \subseteq \mathcal{R}$ such that every element of $\mathcal{X}$ is contained in at least one set of $\mathcal{S}$, and a *hitting set* of Σ is a subset $\mathcal{H} \subseteq \mathcal{X}$ such that each set of $\mathcal{R}$ contains at least one element of $\mathcal{H}$. The goal of the *set cover problem* (respectively, *hitting set problem*) is to find a set cover (respectively, hitting set) of minimum size. In the *weighted version* of the set cover problem (respectively, hitting set problem), the sets in $\mathcal{R}$ (respectively, the elements in $\mathcal{X}$) have some positive weights, and the aim is to obtain a set cover (respectively, hitting set) of minimum weight. Here, the weight of a set means the sum of the weights of all the elements in the set. In the geometric setup, the set $\mathcal{X}$ consists of points in $\mathbb{R}^d$, and the set $\mathcal{R}$ consists of geometric objects (e.g., disks, rectangles, hypercubes, etc.) in $\mathbb{R}^d$. With a slight abuse of the notation, we use $\mathcal{R}$ to signify both the set of $\{\mathcal{X} \cap R \mid R \in \mathcal{R}\}$ as well as the set of objects that define these sets. It is well known that a set cover of $\Sigma = (\mathcal{X}, \mathcal{R})$ is a hitting set of the dual range space $\Sigma^\perp = (\mathcal{X}^\perp, \mathcal{R}^\perp)$. Here, for each range $R \in \mathcal{R}$, there is an element in $\mathcal{X}^\perp$, and for each element $p \in \mathcal{X}$, there is a range R_p, namely $R_p = \{R \in \mathcal{R} \mid p \in R\}$, in $\mathcal{R}^\perp$ [1]. Thus, if there is no restriction on the range space, then both problems are equivalent to each other. In particular, the set cover problem for a range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where $\mathcal{X}$ is a set of points and $\mathcal{R}$ is a set of translated copies of a convex object C is equivalent to the hitting set problem for the dual range space $\Sigma^\perp = (\mathcal{X}^\perp, \mathcal{R}^\perp)$, where $\mathcal{X}^\perp$ consists of centers of the objects in $\mathcal{R}$ and $\mathcal{R}^\perp$ consists of translated copies of the reflected object $-C$ (see Section 2 for the definition reflected object) centered at the points in $\mathcal{X}$ [7] (see Figure 1).



(a) (b)

Figure 1: (a) An instance $\Sigma = (\mathcal{X}, \mathcal{R})$ of the set cover problem such that $\mathcal{X} = \{p_1, \ldots, p_6\}$ and $\mathcal{R} = \{R_1, \ldots, R_5\}$. Here, $\{R_1, R_2, R_5\}$ is a minimum set cover of Σ. (b) An equivalent instance $\Sigma^\perp = (\mathcal{X}^\perp, \mathcal{R}^\perp)$ of the hitting set problem such that $\mathcal{X}^\perp = \{r_i \mid r_i \text{ is center of } R_i \text{ for } 1 \leq i \leq 5\}$ and $\mathcal{R}^\perp = \{P_j \mid P_j \text{ is a unit regular pentagon centered at } p_j \text{ for } 1 \leq j \leq 6\}$. Here, $\{r_1, r_2, r_5\}$ is a minimum hitting set of $\Sigma^\perp$.

Both set cover and hitting set problems belong to Karp's

*Work on this paper by M. De has been partially supported by SERB MATRICS Grant MTR/2021/000584, and work by R. Mandal has been supported by CSIR, India, File Number-09/0086(13712)/2022-EMR-I.

†Dept. of Mathematics, Indian Institute of Technology Delhi, New Delhi, India, minati@maths.iitd.ac.in

‡Dept. of Mathematics, Indian Institute of Technology Delhi, New Delhi, India, maz218522@iitd.ac.in

§Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India, nandysc@isical.ac.in

21 classic NP-hard problems [15]. A basic greedy algorithm achieves $\log(|\mathcal{X}|)$ and $\log(|\mathcal{R}|)$-approximation for the set cover and hitting set problems, respectively [23], which are essentially the best possible under a plausible complexity-theoretic assumption [11]. Both problems remain NP-hard for simple geometric objects such as unit disks and unit squares [12]. Due to Brönnimann and Goodrich [3], a log(OPT)-approximation algorithm is known for the set cover problem for a range space with constant VC-dimension[1], where OPT is the size of an optimum solution. Additionally, both the problems admit PTAS for unit disks [21] and axis-parallel unit squares [13]. In the weighted version of both set cover and hitting set problems, a PTAS is also known for unit disks [16] and unit square [10]. For a range space consisting of points and axis-parallel rectangles with bounded integer side lengths, the weighted set cover problem admits PTAS due to [18]. The best-known result of the (unweighted) hitting set problem for axis-parallel rectangles in $\mathbb{R}^2$ or $\mathbb{R}^3$ is a $\log\log(\text{OPT})$-approximation algorithm due to Aronov et al. [2], where OPT is the size of an optimum hitting set.

However, researchers have studied both set cover and hitting set problems for certain restricted classes of rectangles. Some of them are mentioned below. A rectangle Q in $\mathbb{R}^2$ is said to be a *boundary rectangle* with respect to a rectangular region $\tilde{R}$ if Q lies within $\tilde{R}$ and has exactly one boundary of it attached to a boundary of the rectangular region $\tilde{R}$ (see Figure 1(a)). Chan and Grant [4] showed that both weighted set cover and hitting set problems can be solved in polynomial time for a range space consisting of points, and boundary rectangles anchored on only the bottom boundary of a rectangular region $\tilde{R}$. However, if the rectangles are anchored on only two opposite boundaries of $\tilde{R}$, Mudgal and Pandit [20] showed that both problems are NP-hard. Observe that the later family of boundary rectangles are pseudo-disks, and hence, the existence of PTAS is known for both problems [8, 21]. But, if the boundary rectangles are anchored on any boundary of a rectangular region $\tilde{R}$, then this family of rectangles is not a pseudo-disk. While, for the hitting set problem, a $(2+\epsilon)$-approximation algorithm can be obtained due to [21], there is no known non-trivial approximation algorithm for the set cover problem of boundary rectangles. So, we consider the set cover problem of boundary rectangles and a restricted version of it as follows.

Problem 1: Set Cover Problem of Boundary Rectangles (SC-BR Problem). Set cover problem for a range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where $\mathcal{X} \subseteq \mathbb{R}^2$ is a set of points and $\mathcal{R}$ is a set of boundary rectangles with respect to a rectangular region $\tilde{R}$.

Problem 2: SC-BR-LT Problem. Set cover problem for a range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where $\mathcal{X} \subseteq \mathbb{R}^2$ is a set of points and $\mathcal{R}$ is a set of boundary rectangles with respect to a rectangular region $\tilde{R}$ such that each rectangle in $\mathcal{R}$ attaches to either the left or the top boundary of $\tilde{R}$.

We prove that the SC-BR-LT problem is APX-hard, and so is the SC-BR problem.

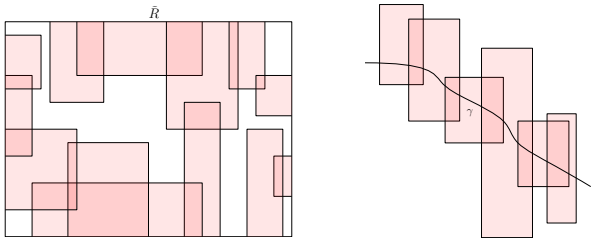


Figure 2: (a) A set of boundary rectangles with respect to a rectangular region $\tilde{R}$. (b) A set of cross-separable rectangles with respect to a x-monotone curve γ.

Chepoi and Felsner [5] studied the hitting set problem for a range space $(\mathcal{X}, \mathcal{R})$, where $\mathcal{X}$ consists of all points in $\mathbb{R}^2$ and $\mathcal{R}$ is a set of rectangles intersecting a x-monotone curve, and provided a 6-approximation algorithm. For the same range space, later, Correa et al. [6] gave an improved 4-approximation algorithm for the same problem. A rectangle Q in $\mathbb{R}^2$ is said to be a *cross-separable rectangle* with respect to a x-monotone curve γ if the curve γ intersects the left and the right boundary of Q (see Figure 1(b)). Due to the result of Chan and Grant [4], and Madireddy and Mudgal [17], both set cover and hitting set problems of cross-separable rectangles are known to be APX-hard, respectively. A 4-approximation algorithm for the hitting set problem is known due to Dey et al [9] for a range space $(\mathcal{X}, \mathcal{R})$, where $\mathcal{X} \subseteq \mathbb{R}^2$ is set of points, and $\mathcal{R}$ is a set of cross-separable rectangles with respect to a x-monotone curve γ. Surprisingly, there is no known constant factor approximation algorithm for the set cover problem for this range space. So, we consider the following problem.

Problem 3: Set Cover Problem of Cross-

[1]For a given range space $\Sigma = (\mathcal{X}, \mathcal{R})$, a subset $\mathcal{Y} \subseteq \mathcal{X}$ is said to be *shattered* by $\mathcal{R}$ if $\mathcal{R}_{|\mathcal{Y}} = \{Q \cap \mathcal{Y} \mid Q \in \mathcal{R}\} = 2^{\mathcal{Y}}$. The range space Σ is said to have a *VC-dimension* d if d is the smallest integer such that no $d+1$ point subset $\mathcal{Y} \subseteq \mathcal{X}$ can be shattered.

Separable Rectangles (SC-CSR Problem). Set cover problem for a range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where $\mathcal{X} \subseteq \mathbb{R}^2$ is a set of points and $\mathcal{R}$ is a set of cross-separable rectangles with respect to a x-monotone curve γ.

When the objects are axis-parallel unit squares touching a diagonal line, Mudgal et al. [19] proved that both (unweighted) set cover and hitting set problems are solvable in polynomial time. In this paper, we generalize their results for a set of translated copies of a convex object touching a diagonal line. We define these problems in the following way.

Problem 4: SC-TCO-DT Problem. Set cover problem for a range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where $\mathcal{X} \subseteq \mathbb{R}^2$ is a set of points and $\mathcal{R}$ is a set of translated copies of a convex object touching a diagonal line from the right side.

Problem 5: HS-TCO-DT Problem. Hitting set problem for a range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where $\mathcal{X} \subseteq \mathbb{R}^2$ is a set of points and $\mathcal{R}$ is a set of translated copies of a convex object touching a diagonal line from the right side.

Our Contributions. This paper primarily considers the set cover and the hitting set problems for some restricted classes of axis-parallel rectangles. In Section 3, we consider the set cover problem of boundary rectangles. Here, we first show that the SC-BR-LT problem is APX-hard by providing a simple encoding of the SPECIAL-3SC problem (see Definition 1) that is already known to be APX-hard [4]. Then, we present a 4 and a 8-approximation algorithm for the SC-BR-LT problem and the SC-BR problem, respectively. Furthermore, in Section 4, we consider the weighted SC-CSR problem which is an APX-hard problem due to the result of Chan and Grant [4, Theorem 1.1(C1)]. For this problem, we obtain a 2-approximation algorithm. Finally, in Section 5, we extend the result of [19] to show that the weighted SC-TCO-DT and HS-TCO-DT problems can be solved in polynomial time.

2 Notation and Preliminaries

Throughout the paper, points are denoted by small letters, objects by capital letters, and sets of points/objects by calligraphic font. We use $[n]$ to represent the set $\{1, \ldots, n\}$ for a positive integer n. Let $Q = \square q_1 q_2 q_3 q_4$ be a rectangle. The boundaries q_1q_2, q_2q_3, q_3q_4, and q_4q_1 will be called the left, the bottom, the right, and the top boundary of Q, respectively. A point p is said to be *contained* by a rectangle Q (or Q *contains* p) if $p \in Q$.

A point p' is said to be a *reflection of a point* p in a plane through a point q if the midpoint of the line segment $\overline{pp'}$ is q. Now, we define the *reflection of a convex object* C through a point p, denoted by $-C(p)$, as the collection of reflection of all points in C through the point p. We use the term $-C$ to denote a translated copy of $-C(p)$. We use the term *center* of a convex object C to mean the center of the smallest disk circumscribing the object C. Now, we have the following observation (see Figure 3).

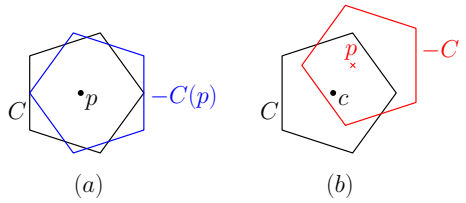


Figure 3: (a) Here, C (black) is a convex object, and the object $-C(p)$ (blue) is a reflected copy of C through a point p. (b) Illustration of Observation 1.

Observation 1 *[7] Let C be a convex object centered at a point c, and $-C$ be a reflected copy of C centered at a point p. Then, the point p lies in the convex object C if and only if the reflected object $-C$ contains the point c.*

3 Set Cover Problem of Boundary Rectangles

In this section, we consider the set cover problem of boundary rectangles (SC-BR Problem) and its two restricted versions, SC-BR-LT problem and SC-BR-B problem, where in the SC-BR-B problem, the objects are boundary rectangles with respect to a rectangular region $\tilde{R}$ anchored on the bottom boundary of $\tilde{R}$. Theorem 1 suggests a polynomial time algorithm for the weighted SC-BR-B problem.

Theorem 1 *The weighted SC-BR-B problem can be solved in $O(mn^3 \log^2 n)$-time.*

Proof. Let $\Sigma = (\mathcal{X}, \mathcal{R})$ be an instance of an SC-BR-B problem such that $\mathcal{X} = \{p_1, \ldots, p_n\}$ and $\mathcal{R} = \{R_1, \ldots, R_m\}$, where $\mathcal{R}$ is a set of boundary rectangles with respect to a rectangular region $\tilde{R}$ such that every rectangle in $\mathcal{R}$ attaches to the bottom boundary of $\tilde{R}$. Also, for each $R \in \mathcal{R}$, let $w(R)$ denote the weight of the rectangle R. W.l.o.g., assume that each point in $\mathcal{X}$ is contained in at least two rectangles of $\mathcal{R}$.

Consider two vertical lines l_1 and l_2. Let $\mathcal{X}(l_1, l_2)$ (respectively, $\mathcal{R}(l_1, l_2)$) be the set of points (respectively, rectangles) lying fully inside the vertical strip

formed by the lines l_1 and l_2. Also, suppose that $\mathcal{S}(l_1, l_2)$ be an optimum set cover of the instance $\Sigma_{(l_1,l_2)} = (\mathcal{X}(l_1, l_2), \mathcal{R}(l_1, l_2))$ and $\text{OPT}(l_1, l_2) = \sum_{R\in\mathcal{S}(l_1,l_2)} w(R)$. Now, we have the following two properties.

Property 1. Each rectangle in $\mathcal{S}(l_1, l_2)$ contains a point of $\mathcal{X}(l_1, l_2)$ that is not contained in any other rectangle of $\mathcal{S}(l_1, l_2)$.

Property 2. Let $R_{\min}$ be a rectangle in $\mathcal{S}(l_1, l_2)$ such that the top boundary of $R_{\min}$ lies below the top boundary of each rectangle in $\mathcal{S}(l_1, l_2)$. Also, let $p \in \mathcal{X}(l_1, l_2)$ be the point uniquely contained in $R_{\min}$ (due to Property 1). Then, any rectangle of $\mathcal{S}(l_1, l_2)$ does not intersect the vertical line $l(p)$ passing through the point p.

Due to the above two properties, we have the following lemma.

Lemma 2 *For the range space $\Sigma_{(l_1,l_2)} = (\mathcal{X}(l_1, l_2), \mathcal{R}(l_1, l_2))$, let $\mathcal{S}(l_1, l_2)$ be an optimum set cover of $\Sigma_{(l_1,l_2)}$, and $OPT(l_1, l_2) = \sum_{R\in\mathcal{S}(l_1,l_2)} w(R)$ where the weight of a rectangle R is denoted by $w(R)$. Then, we have*

$$OPT(l_1, l_2) = \min_{R\in\mathcal{R}(l_1,l_2)} \{w(R) + \min_{p\in R\cap\mathcal{X}(l_1,l_2)} \{OPT(l_1, l(p)) + OPT(l(p), l_2)\}\}.$$

Let $p \in R \cap \mathcal{X}(l_1, l_2)$ and $R \in \mathcal{R}(l_1, l_2)$ be such that $OPT(l_1, l_2) = w(R) + OPT(l_1, l(p)) + OPT(l(p), l_2)$. Then, we have

$$S(l_1, l_2) = \{R\} \cup \mathcal{S}(l_1, l(p)) \cup \mathcal{S}(l(p), l_2).$$

Observe that the optimum set cover $\mathcal{S}$ of the range space $\Sigma = (\mathcal{X}, \mathcal{R})$ is equal to $\mathcal{S}(l', l'')$, where l' and l'' are the vertical lines passing through the left and right boundary of $\tilde{R}$, respectively. So, $\mathcal{S} = \mathcal{S}(l', l'')$ and $\text{OPT}(l', l'') = \sum_{R\in\mathcal{S}} w(R)$. Because of this, Lemma 2 immediately implies that there is dynamic programming to obtain $\mathcal{S}(l', l'')$ and $\text{OPT}(l', l'')$. Now, the total number of primary subproblems (i.e. $\Sigma_{(l_1,l_2)}$) is $O(n^2)$ as each vertical line passes through a point. Further, for each such primary subproblem, we have another $O(mn)$ subproblem as we need to consider each rectangle $R \in \mathcal{R}(l_1, l_2)$ and for each such rectangle, we also need to consider each point $p \in R \cap \mathcal{X}(l_1, l_2)$ inside it. Also, we need another $O(\log^2 n)$-time to calculate $\text{OPT}(l_1, l(p))$ and $\text{OPT}(l(p), l_2)$ by using a 2D range tree. □

Now, we focus on the SC-BR-LT problem. Using an encoding of the SPECIAL-3SC problem, stated below, we prove that the problem is APX-hard.

Definition 1 (SPECIAL-3SC Problem [4]) *We are given a universe $\mathcal{U} = \mathcal{A}\cup\mathcal{W}\cup\mathcal{X}\cup\mathcal{Y}\cup\mathcal{Z}$ comprising disjoint sets $\mathcal{A} = \{a_1, \ldots, a_n\}$, $\mathcal{W} = \{w_1, \ldots, w_m\}$, $\mathcal{X} = \{x_1, \ldots, x_m\}$, $\mathcal{Y} = \{y_1, \ldots, y_m\}$ and $\mathcal{Z} = \{z_1, \ldots, z_m\}$, where $2n = 3m$. We are also given a family $\mathcal{S}$ of $5m$ subsets of $\mathcal{U}$ satisfying the following two conditions:*

1. For each $1 \le t \le m$, there are integers $1 \le i < j < k \le n$ such that $\mathcal{S}$ contains the sets $\{a_i, w_t\}$, $\{w_t, x_t\}$, $\{x_t, a_j, y_t\}$, $\{y_t, z_t\}$ and $\{z_t, a_k\}$ (summing over all t gives the $5m$ sets contained in $\mathcal{S}$).

2. For all $1 \le l \le n$, the element a_l is in exactly two sets in $\mathcal{S}$.

The SPECIAL-3SC problem denotes the set cover problem on the range space $\Sigma = (\mathcal{U}, \mathcal{S})$.

Chan and Grant [4] showed the APX-hardness proof of the SPECIAL-3SC problem by providing a reduction from the minimum vertex cover problem of the 3-regular graph. To prove the APX-hardness result of the SC-BR-LT problem, we construct a point for each element of $\mathcal{U}$ and a boundary rectangle for each set of $\mathcal{S}$.

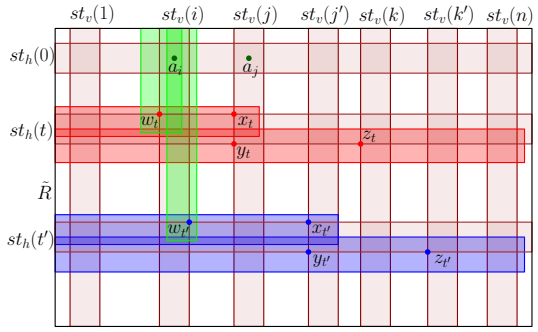


Figure 4: Here, $1 \le i < j < j' < k < k' \le n$ and $1 \le t < t' \le m$. The family $\mathcal{S}$ contains the sets $\{a_i, w_t\}$, $\{w_t, x_t\}$, $\{x_t, a_j, y_t\}$, $\{y_t, z_t\}$, $\{z_t, a_k\}$, $\{a_i, w_{t'}\}$, $\{w_{t'}, x_{t'}\}$, $\{x_{t'}, a_{j'}, y_{t'}\}$, $\{y_{t'}, z_{t'}\}$ and $\{z_{t'}, a_{k'}\}$.

Encoding the SPECIAL-3SC Problem to the SC-BR-LT Problem. Consider a rectangular region $\tilde{R}$. See Figure 4. For each $i \in [n]$, we consider a vertical strip, say $st_v(i)$, inside $\tilde{R}$ such that for $1 \le i < j \le n$, the strip $st_v(i)$ lies on the left side of the strip $st_v(j)$, and $st_v(i) \cap st_v(j) = \emptyset$. Next, for each $0 \le t \le m$, we consider a horizontal strip $st_h(t)$ such that for $0 \le$

$t < t' \leq m$, the strip $st_h(t)$ lies above the strip $st_h(t')$, and $st_h(t) \cap st_h(t') = \emptyset$. We use $cell(i, j)$ to denote the intersection region of the strips $st_v(i)$ and $st_h(j)$. Now, for each $i \in [n]$, we place the point a_i at the middle of $cell(0, i)$. Now, for each $1 \leq t \leq m$, we place points w_t, x_t, y_t, z_t on the intersection points of the boundary of the horizontal strip $st_h(t)$ with some vertical strips as follows. Considering Condition 1 of Definition 1, let $1 \leq i < j < k \leq n$ be such that $\mathcal{S}$ contains the sets $\{a_i, w_t\}$, $\{w_t, x_t\}$, $\{x_t, a_j, y_t\}$, $\{y_t, z_t\}$ and $\{z_t, a_k\}$. We place w_t (resp., x_t) on one of the corners of the top boundary of $cell(t, i)$ (resp., $cell(t, j)$). Whereas y_t (resp., z_t) is placed on one of the (left or right) corners of the bottom boundary of $cell(t, j)$ (resp., $cell(t, k)$). Since there are no other points on the top boundary of $st_h(t)$ apart from w_t and x_t, we can draw a boundary rectangle anchored on the left boundary of $\tilde{R}$ containing only w_t and x_t as shown in Figure 4. Similarly, we draw a boundary rectangle anchored on the left boundary of $\tilde{R}$ containing only y_t and z_t. For each of the sets involving elements from the set $\mathcal{A}$, we draw a boundary rectangle anchored on the top boundary of $\tilde{R}$. Due to Condition 2 of Definition 1, we know that each $a_i \in \mathcal{A}$ is in exactly two sets in $\mathcal{S}$. To ensure that we can draw two rectangles sharing only a_i but no other points, we do the following. While deciding the corner (left or right) for the placement of the point w_t (resp., x_t and z_t), if there is no point on the left boundary of the strip $st_v(i)$ (resp., $st_v(j)$ and $st_v(k)$), we place the point on the left, otherwise at the right corner. Since x_t and y_t are the corners of the same $cell(t, j)$, we set the left/right placement decision of y_t the same as that for x_t. This convention guarantees that we can draw three rectangles corresponding to $\{a_i, w_t\}$, $\{x_t, a_j, y_t\}$ and $\{z_t, a_k\}$ (as shown in Figure 4). Therefore, we have the following theorem.

Theorem 3 *The SC-BR-LT problem is APX-hard.*

Now, we present an 8-approximation algorithm for the SC-BR problem. Let $\Sigma = (\mathcal{X}, \mathcal{R})$ be an instance of the SC-BR problem such that $\mathcal{X} = \{p_1, \ldots, p_n\}$ and $\mathcal{R} = \{R_1, \ldots, R_m\}$, where $\mathcal{R}$ is a set of boundary rectangles with respect to a rectangular region $\tilde{R}$.

First, we define the set $\mathcal{R}^{(1)} = \{R_i \in \mathcal{R} \mid$ bottom boundary of R_i attaches to the bottom boundary of $\tilde{R}\}$. Similarly, we can define the other subsets $\mathcal{R}^{(2)}, \mathcal{R}^{(3)}$ and $\mathcal{R}^{(4)}$ corresponding to the right, top and left boundary of $\tilde{R}$, respectively. Now, for each rectangle $R_i \in \mathcal{R}^{(t)}$, we consider a binary variable $x_i^{(t)}$, where $t \in [4]$. For a specific solution, the value of $x_i^{(t)}$ is 1 or 0 depending on whether the square $R_i \in \mathcal{R}^{(t)}$ is in the solution or not, respectively. The ILP formulation P_1 of the SC-BR problem for the instance $\Sigma = (\mathcal{X}, \mathcal{R})$ is as follows.

$$\mathrm{P}_1 : \min \sum_{t \in [4]} \sum_{i | R_i \in \mathcal{R}^{(t)}} x_i^{(t)}$$
$$\text{subject to,} \sum_{t \in [4]} \sum_{i | p \in R_i \in \mathcal{R}^{(t)}} x_i^{(t)} \geq 1 \ \forall p \in \mathcal{X}$$
$$x_i^{(t)} \in \{0, 1\} \ \forall i \text{ s.t. } R_i \in \mathcal{R}^{(t)} \text{ for } t \in [4]$$

Let $\overline{\mathrm{P}}_1$ be the relaxed LP problem corresponding to the ILP problem P_1. Now, consider an optimum solution $\overline{x}$ of the problem $\overline{\mathrm{P}}_1$. Next, we partition the point set $\mathcal{X}$ into four sets $\mathcal{X}^{(1)}$, $\mathcal{X}^{(2)}$, $\mathcal{X}^{(3)}$ and $\mathcal{X}^{(4)}$ corresponding to the optimum solution $\overline{x}$ as follows. For $t \in [4]$, we consider $\mathcal{X}^{(t)} = \{p \in \mathcal{X} \mid \sum_{i | p \in R_i \in \mathcal{R}^{(t)}} \overline{x}_i^{(t)} \geq \frac{1}{4}\}$.

For a point $p \in \mathcal{X}$, if more than one of the above four conditions holds, then we put p arbitrarily in any one of the four sets. Note that each of the range spaces $(\mathcal{X}^{(t)}, \mathcal{R}^{(t)})$ for $t \in [4]$ can be considered as an instance of the weighted SC-BR-B problem, where the weight of each rectangle in $\mathcal{R}^{(t)}$ is 1. So, we solve each of the instances $(\mathcal{X}^{(t)}, \mathcal{R}^{(t)})$ for $t \in [4]$ independently, using Theorem 1, and output their union as a solution of the instance $(\mathcal{X}, \mathcal{R})$. Let $\mathrm{OPT}^{(t)}(I)$ be the optimum set cover of the instance $(\mathcal{X}^{(t)}, \mathcal{R}^{(t)})$ for $t \in [4]$ obtained by using Theorem 1. We set $\mathrm{ALG} = \cup_{t=1}^{4} \mathrm{OPT}^{(t)}(I)$, which is a set cover of the instance $(\mathcal{X}, \mathcal{R})$.

The ILP formulation $\mathrm{P}_2^{(t)}$ of the SC-BR-B problem for the instance $(\mathcal{X}^{(t)}, \mathcal{R}^{(t)})$, where $t \in [4]$, as follows.

$$\mathrm{P}_2^{(t)} : \min \sum_{i | R_i \in \mathcal{R}^{(t)}} x_i^{(t)}$$
$$\text{subject to,} \sum_{i | p \in R_i \in \mathcal{R}^{(t)}} x_i^{(t)} \geq 1 \ \forall p \in \mathcal{X}^{(t)}$$
$$x_i^{(t)} \in \{0, 1\} \ \forall i \text{ s.t. } R_i \in \mathcal{R}^{(t)}$$

For $t \in [4]$, let $\overline{\mathrm{P}}_2^{(t)}$ be the relaxed LP problem corresponding to the ILP problem $\mathrm{P}_2^{(t)}$. Let $\mathrm{OPT}_2^{(t)}(I)$ and $\mathrm{OPT}_2^{(t)}(F)$ be the optimum solution of $\mathrm{P}_2^{(t)}$ and $\overline{\mathrm{P}}_2^{(t)}$, respectively, for $t \in [4]$. Now, we have the following lemma.

Lemma 4 *$OPT_2^{(t)}(I) \leq 2 OPT_2^{(t)}(F)$ for $t \in [4]$.*

Proof. We proof the lemma only for $t = 1$. Other cases are similar in nature. First, we define an algorithm, say Algorithm 1, for the set cover problem of the range space $(\mathcal{X}^{(1)}, \mathcal{R}^{(1)})$ which uses two sets $\mathcal{X}'$ and $\mathcal{R}'$ such

that both are empty at the beginning. Let p be a point in $\mathcal{X}^{(1)}$ with maximum y-coordinate, and $\mathcal{R}_p$ be the set of all rectangles in $\mathcal{R}^{(1)}$ that contains p. Also, let R_l (respectively, R_r) be the rectangle in $\mathcal{R}_p$ having a left-most left boundary (respectively, right-most right boundary.). Now, first, Algorithm 1 updates $\mathcal{X}'$ and $\mathcal{R}'$ by adding p in $\mathcal{X}'$, and the rectangles R_l, R_r in $\mathcal{R}'$. Suppose, $\mathcal{R}_1 = \mathcal{R}^{(1)} \setminus \mathcal{R}_p$ and $\mathcal{X}_1 = \mathcal{X}^{(1)} \setminus \mathcal{X}_p$, where $\mathcal{X}_p$ consists of all the points of $\mathcal{X}^{(1)}$ that are contained in any rectangle of $\mathcal{R}_p$. Observe that any points in $\mathcal{X}_p$ are covered by $\mathcal{R}_l \cup \mathcal{R}_r$. Next, if the set $\mathcal{X}_1$ is not empty, Algorithm 1 will do the same procedure for the range space $(\mathcal{X}_1, \mathcal{R}_1)$, recursively. At last, it will output $\mathcal{R}'$. The pseudo-code of the algorithm is given below.

Algorithm 1 An Approximation Algorithm for the SC-BR-B Problem ScBrB$(\mathcal{X}^{(1)}, \mathcal{R}^{(1)})$

1: $p \leftarrow$ the point in $\mathcal{X}^{(1)}$ with maximum y-coordinate.
2: $\mathcal{R}_p \leftarrow$ the set of all rectangles in $\mathcal{R}^{(1)}$ that contains p.
3: $R_l \leftarrow$ the rectangle in $\mathcal{R}_p$ having left-most left boundary.
4: $R_r \leftarrow$ the rectangle in $\mathcal{R}_p$ having right-most right boundary.
5: Put $p \in \mathcal{X}'$ and $R_l, R_r \in \mathcal{R}'$.
6: Let $\mathcal{R}_1 = \mathcal{R}^{(1)} \setminus \mathcal{R}_p$ and $\mathcal{X}_1 = \mathcal{X}^{(1)} \setminus \mathcal{X}_p$, where $\mathcal{X}_p$ consists of all the points of $\mathcal{X}^{(1)}$ that are contained in $\mathcal{R}_p$.
7: **if** $\mathcal{X}_1 \neq \emptyset$ **then**
8: $\quad \mathcal{R}' \leftarrow$ ScBrB$(\mathcal{X}_1, \mathcal{R}_1) \cup \mathcal{R}'$.
9: **end if**
10: Report $\mathcal{R}'$.

Recall that $\mathcal{X}'$ consists of all those points corresponding to which Algorithm 1 adds two rectangles in $\mathcal{R}'$. We now show that $|\mathcal{X}'| \leq \mathrm{OPT}_2^{(1)}(F)$. Observe that for each pair of points p and q in $\mathcal{X}'$, there does not exist a rectangle in $\mathcal{R}^{(1)}$ that contains both the points p and q. As a result, the variables that appear in the constraints corresponding to p do not appear in the constraints of q. Hence, $|\mathcal{X}'| \leq \mathrm{OPT}_2^{(1)}(F)$. Since for each point $p \in \mathcal{X}'$, Algorithm 1 puts two square in $\mathcal{R}'$, we have that $|\mathcal{R}'| \leq 2|\mathcal{X}'|$. Also, note that $\mathcal{R}'$ is a set cover for the range space $(\mathcal{X}^{(1)}, \mathcal{R}^{(1)})$. As a result, $\mathrm{OPT}_2^{(1)}(I) \leq |\mathcal{R}'| \leq 2|\mathcal{X}'| \leq 2\mathrm{OPT}_2^{(1)}(F)$. □

Recall that $\overline{x}$ is an optimal solution to the problem $\overline{\mathrm{P}}_1$. Then, for $t \in [4]$, we have that $4\overline{x}^{(t)}$ satisfies all the constraints of the problem $\overline{\mathrm{P}}_2^{(t)}$. As a result, $\mathrm{OPT}_2^{(t)}(F) \leq 4\sum_i \overline{x}_i^{(t)}$. Let $\mathrm{OPT}_1(I)$ and $\mathrm{OPT}_1(F)$ be the optimum solution of P_1 and $\overline{\mathrm{P}}_1$, respectively. Note that $\mathrm{OPT}_1(F) = \sum_{t,i} \overline{x}_i^{(t)}$. So, $\sum_{t=1}^4 \mathrm{OPT}_2^{(t)}(F) \leq 4\mathrm{OPT}_1(F) \leq 4\mathrm{OPT}_1(I)$. Now, we have that $|\mathrm{ALG}| = \sum_{t=1}^4 |\mathrm{OPT}^{(t)}(I)| \leq 2\sum_{t=1}^4 \mathrm{OPT}_2^{(t)}(F) \leq 2 \cdot 4\mathrm{OPT}_1(I) = 8\mathrm{OPT}_1(I)$ (here, second and third inequality are due to Lemma 4, and $\sum_{t=1}^4 \mathrm{OPT}^{(t)}(F) \leq 4\mathrm{OPT}_1(I)$, respectively).

The running time of our algorithm depends on solving the LP problem $\overline{\mathrm{P}}_1$ as well as obtaining a set cover for each of the range spaces $(\mathcal{X}^{(t)}, \mathcal{R}^{(t)})$ by using Theorem 1. Hence, the running time of our algorithm is polynomial in n and m. Therefore, we have the following theorem.

Theorem 5 *There exists a polynomial time 8-approximation algorithm for the SC-BR problem.*

Observe that our algorithm is also a 4-approximation algorithm for the SC-BR-LT problem since the two sets $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$ are empty.

4 Approximation Algorithm for the Weighted SC-CSR Problem

In this section, we present a 2-approximation algorithm for the weighted SC-CSR problem. Let $\Sigma = (\mathcal{X}, \mathcal{R})$ be an instance of an SC-CSR problem such that $\mathcal{X} = \{p_1, \ldots, p_n\}$ and $\mathcal{R} = \{R_1, \ldots, R_m\}$, where $\mathcal{R}$ is a set of cross-separable rectangles with respect to a x-monotone curve γ. For each $R \in \mathcal{R}$, let $w(R)$ be the weight of the rectangle R. W.l.o.g., we assume that no point in $\mathcal{X}$ lies on the curve γ.

Let $\mathcal{X}_1 = \{p_j \in \mathcal{X} \mid$ the point p_j lies above the curve $\gamma\}$ and $\mathcal{X}_2 = \{p_j \in \mathcal{X} \mid$ the point p_j lies below the curve $\gamma\}$ be a partition of the set $\mathcal{X}$. Also, $\mathcal{R}_1 = \{R_i \in \mathcal{R} \mid R_i$ contains at least a point of $\mathcal{X}_1\}$ and $\mathcal{R}_2 = \{R_i \in \mathcal{R} \mid R_i$ contains at least a point of $\mathcal{X}_2\}$. Thus, we have two range spaces, namely $\Sigma_1 = (\mathcal{X}_1, \mathcal{R}_1)$ and $\Sigma_2 = (\mathcal{X}_2, \mathcal{R}_2)$.

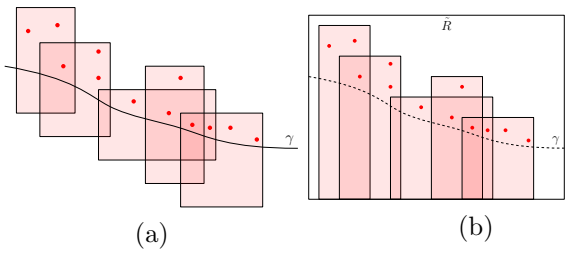


Figure 5: (a) A example of an instance of the range space $\Sigma_1 = (\mathcal{X}_1, \mathcal{R}_1)$. (b) An equivalent instance of the SC-BR-B problem.

Now, $\Sigma_1 = (\mathcal{X}_1, \mathcal{R}_1)$ is an instance of the weighted SC-CSR problem such that all points of $\mathcal{X}_1$ lie above the curve γ. As a result of these, the range space $\Sigma_1 = (\mathcal{X}_1, \mathcal{R}_1)$ can be considered as an instance of the weighted SC-BR-B problem (see Figure 5). Therefore, we can obtain an optimum set cover for Σ_1 in polynomial time by using Theorem 1. Similarly, we can obtain an optimum set cover for the range space Σ_2 in polynomial time by using Theorem 1.

Let ALG_1 and ALG_2 be an optimum set cover of $\Sigma_1 = (\mathcal{X}_1, \mathcal{R}_1)$ and $\Sigma_2 = (\mathcal{X}_2, \mathcal{R}_2)$, respectively, obtained by using Theorem 1. Consider $\text{ALG} = \text{ALG}_1 \cup \text{ALG}_2$. Let OPT be an optimum solution of Σ. Again, let $\text{OPT}_1 = \{R \in \text{OPT} \mid R \in \mathcal{R}_1\}$ and $\text{OPT}_2 = \{R \in \text{OPT} \mid R \in \mathcal{R}_2\}$. Let $w(\text{ALG}), w(\text{ALG}_1), w(\text{ALG}_2)$ and $w(\text{OPT})$ be the sum of the weight of all rectangles in $\text{ALG}, \text{ALG}_1, \text{ALG}_2$ and OPT, respectively. It is easy to observe that OPT_t is a set cover of the range space Σ_t for $t = 1, 2$. As a result, $w(\text{ALG}_t) \leq w(\text{OPT}_t)$ for $t = 1, 2$. So, we have that $w(\text{ALG}) = \sum_{t=1}^{2} w(\text{ALG}_t) \leq \sum_{t=1}^{2} w(\text{OPT}_t) \leq 2w(\text{OPT})$.

We need $O(mn)$-time to construct the four sets $\mathcal{X}_t$ and $\mathcal{R}_t$ for $t \in [2]$. Also, in $O(mn^3 \log^2 n)$-time, we get the optimum set cover for the range space Σ_t due to Theorem 1. Therefore, we have the following theorem.

Theorem 6 *There exists a 2-approximation algorithm for the weighted SC-CSR problem with running time $O(mn^3 \log^2 n)$.*

5 Polynomial Time Algorithm for the Weighted SC-TCO-DT and HS-TCO-DT Problems

In this section, we consider the weighted SC-TCO-DT and HS-TCO-DT problems and provide a polynomial time algorithm for both of them. Let C be a convex object in $\mathbb{R}^2$. Let us consider the weighted SC-TCO-DT problem for the range space $\Sigma = (\mathcal{X}, \mathcal{R})$, where $\mathcal{X} = \{p_1, \ldots, p_n\}$ is a set of n points and $\mathcal{R} = \{R_1, \ldots, R_m\}$ is a set of m translated copies of C touching a diagonal line l from the right side (see Figure 6(a)). We assume that each rectangle $R_j \in \mathcal{R}$ has a weight $w(R_j)$. Let l' be a line parallel to l passing through the centers of all the objects of $\mathcal{R}$ (see Figure 6(b)). Let $\mathcal{X}' = \{c_j \mid c_j \text{ is the center of } R_j \text{ for } j \in [m]\}$. For each $c_j \in \mathcal{X}'$, we set the weight $w(c_j) = w(R_j)$. Let $\mathcal{R}' = \{-C_i \cap l' \mid -C_i \text{ is a translated copy of } -C \text{ centered at the point } p_i \text{ for } i \in [n]\}$. Recall that $-C$ is the reflected copy of C (see Section 2). Now, consider the following lemma.

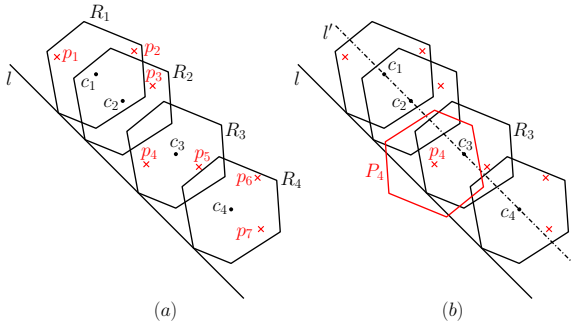


Figure 6: (a) Here, $\mathcal{X} = \{p_1, \ldots, p_7\}$ and $\mathcal{R} = \{R_1, \ldots, R_4\}$ such that each R_j touching a diagonal line l from the right side. (b) The line l' is parallel to l passing through all the centers of the objects in $\mathcal{R}$. Also, R_3 contains the point p_4 and the objects P_4, the reflected copy of R_3 through the point p_4 contains the center c_3 of R_3.

Lemma 7 *For each $1 \leq i \leq n$, the object $-C_i \cap l'$ is an interval on the line l'.*

Proof. Since $-C_i$ is a convex object, $-C_i$ intersects l' either at one point or at two points. If $-C_i \cap l' = \{a\}$, a single point, then we are done. Otherwise, $-C_i \cap l' = \{a, b\}$ i.e. $-C_i$ intersects l' at the points a and b. Since $-C_i$ is a convex object, the line segment joining a and b, say l_{ab}, lies completely inside $-C_i$. As a result, $-C_i \cap l' = l_{ab}$, an interval $[a, b]$ on the line l'. □

Now, due to Observation 1 and Lemma 7, one can prove the following lemma.

Lemma 8 *For $1 \leq i \leq n$ and $1 \leq j \leq m$, a point p_i is contained inside an object R_j if and only if the interval $-C_i \cap l'$ contains the point c_j.*

As a result, the weighted set cover problem for the instance $\Sigma = (\mathcal{X}, \mathcal{R})$ is equivalent to the weighted hitting set problem for the instance $\Sigma' = (\mathcal{X}', \mathcal{R}')$. Similarly, the weighted hitting set problem for the instance $\Sigma = (\mathcal{X}, \mathcal{R})$, with each $p_i \in \mathcal{X}$ having a weight $w(p_i)$, can be reduced to an equivalent weighted set cover problem for the instance $\Sigma' = (\mathcal{X}', \mathcal{R}')$, with each $-C_i \cap l' \in \mathcal{R}'$ having weight $w(-C_i \cap l') = w(p_i)$. Since both weighted set cover and hitting set problems for points and intervals can be solved in $O(nm)$-time [22], we have the following theorem.

Theorem 9 *Both the weighted SC-TCO-DT problem and the weighted HS-TCO-DT problem can be solved in $O(nm)$ time.*

6 Conclusion

Prior to our result, only the hardness results were known for the SC-BR problem and the SC-CSR problem; there were no known constant factors approximation algorithms. Our result provides the first constant factor approximation algorithms for them. Improving the approximation ratio further would be an open problem. For objects touching a diagonal line, no result was known when objects are anything apart from unit squares. We obtain the first polynomial time algorithm for both set cover and hitting set problems when objects are translated copies of a convex object touching a diagonal line.

References

[1] P. K. Agarwal and J. Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discret. Comput. Geom.*, 63(2):460–482, 2020.

[2] B. Aronov, E. Ezra, and M. Sharir. Small-size ϵ-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39(7):3248–3282, 2010.

[3] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discret. Comput. Geom.*, 14(4):463–479, 1995.

[4] 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.

[5] V. Chepoi and S. Felsner. Approximating hitting sets of axis-parallel rectangles intersecting a monotone curve. *Comput. Geom.*, 46(9):1036–1041, 2013.

[6] J. R. Correa, L. Feuilloley, P. Pérez-Lantero, and J. A. Soto. Independent and hitting sets of rectangles intersecting a diagonal line: Algorithms and complexity. *Discret. Comput. Geom.*, 53(2):344–365, 2015.

[7] M. De, S. Jain, S. V. Kallepalli, and S. Singh. Online geometric covering and piercing. *CoRR*, abs/2305.02445, 2023.

[8] M. De and A. Lahiri. Geometric dominating-set and set-cover via local-search. *Comput. Geom.*, 113:102007, 2023.

[9] S. Dey, F. Foucaud, S. C. Nandy, and A. Sen. Complexity and approximation for discriminating and identifying code problems in geometric setups. *Algorithmica*, 85(7):1850–1882, 2023.

[10] T. Erlebach and E. J. van Leeuwen. PTAS for weighted set cover on unit squares. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, September 1-3, 2010. Proceedings*, volume 6302 of *Lecture Notes in Computer Science*, pages 166–177. Springer, 2010.

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

[12] R. J. Fowler, M. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981.

[13] T. F. Gonzalez. Covering a set of points in multidimensional space. *Inf. Process. Lett.*, 40(4):181–188, 1991.

[14] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985.

[15] R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[16] J. Li and Y. Jin. A PTAS for the weighted unit disk cover problem. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 898–909. Springer, 2015.

[17] R. R. Madireddy and A. Mudgal. Approximability and hardness of geometric hitting set with axis-parallel rectangles. *Inf. Process. Lett.*, 141:9–15, 2019.

[18] R. R. Madireddy and A. Mudgal. Weighted geometric set cover with rectangles of bounded integer side lengths. *Discret. Appl. Math.*, 315:36–55, 2022.

[19] A. Mudgal and S. Pandit. Covering, hitting, piercing and packing rectangles intersecting an inclined line. In *Combinatorial Optimization and Applications - 9th International Conference, COCOA 2015, December 18-20, 2015, Proceedings*, volume 9486 of *Lecture Notes in Computer Science*, pages 126–137. Springer, 2015.

[20] A. Mudgal and S. Pandit. Geometric hitting set, set cover and generalized class cover problems with half-strips in opposite directions. *Discret. Appl. Math.*, 211:143–162, 2016.

[21] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discret. Comput. Geom.*, 44(4):883–895, 2010.

[22] D. K. R., A. B. Roy, M. De, and S. Govindarajan. Demand hitting and covering of intervals. In *Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, February 16-18, 2017, Proceedings*, volume 10156 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2017.

[23] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Building Discrete Self-Similar Fractals in Seeded Tile Automata*

Ryan Knobel† Adrian Salinas† Robert Schweller† Tim Wylie†

Abstract

In this paper, we show that a special class of discrete self-similar fractals is *strictly* self-assembled (without error) in the seeded growth-only (no detachments) Tile Automata model. Additionally, we show that under a more restrictive version of the problem, the same class of discrete self-similar fractals is also *super-strictly* buildable– where there is the added requirement of reaching certain intermediate assemblies as the assembly grows. This contrasts with known impossibility results for the abstract Tile Assembly Model, paving the way for future work in strictly self-assembling any generalized discrete self-similar fractal.

1 Introduction

The essence of many organisms and processes of nature can often be described as a collection of simpler, self-organizing components working together to form more complex structures. The study of such mechanisms has resulted in numerous advances in designing artificial programmable systems that accomplish similar tasks. In [12], Winfree introduced the abstract Tile Assembly Model (aTAM), in which single non-rotating 'tiles' attach to growing structures. Other extensions to this model include the 2-Handed Assembly Model (2HAM) [8], where two assemblies are allowed to attach; the Signal-passing Tile Assembly model (STAM) [7], where glues can turn 'on' and 'off' and assemblies can detach; and the seeded Tile Automata Model (seeded TA) [1], where single tiles attach to a base assembly (seed) and adjacent tiles are allowed to change states. While mostly theoretical, experiments realized in the aTAM prove the potential of these programmable systems to build complex structures [10, 11, 12].

Despite varying nuances between models, building precise shapes remains a fundamental task. In particular, one of the most well-studied problems among these models is the self-assembly of self-similar fractals. In [6, 9], it was shown that the aTAM can not strictly (without error) build certain types of self-similar fractals. However, in other models, this does not hold true. In [3], it was shown that the 2HAM can finitely self-assemble a scaled-up Sierpinski carpet, while [5] showed that the 2HAM can finitely self-assemble a larger class of discrete self-similar fractals. In [7], it was shown that the Sierpinski triangle could strictly self-assemble in the STAM if tile detachments are allowed, with [4] providing constructions for any arbitrary discrete self-similar fractal with such detachments, while without such detachments, the finite number of times a STAM tile can change state makes some fractals impossible to build. The STAM is also capable of simulating Tile Automata [2] meaning these results can be ported to the STAM, however, the simulation uses detachments, which is a known result.

In this paper, we focus on building fractals in the seeded TA model without tile detachment, a model differing from the aTAM by the ability for adjacent tiles to transition states. Particularly, we show that a special class of discrete self-similar fractals can be super-strictly built (a more restricted version of strict), leaving a full treatment for future work. Super-strict assembly of a fractal essentially requires that each stage of the fractal be built in order on the way to building the infinite fractal. We feel this is a natural property to strive for as it implies that any intermediate stage of the assembly process would represent precisely the transition between two consecutive stages of the fractal, whereas without, an intermediate assembly could potentially contain a mishmash of many different incomplete fractal stages.

2 Preliminaries

This section defines the model, discrete self-similar fractals, and strictly building shapes as defined in [1, 9].

Seeded Tile Automata. Let Σ denote a set of *states* or symbols. A tile $t = (\sigma, p)$ is a non-rotatable unit square placed at point $p \in \mathbb{Z}^2$ and has a state of $\sigma \in \Sigma$. An *affinity function* Π over a set of states Σ takes an ordered pair of states $(\sigma_1, \sigma_2) \in \Sigma \times \Sigma$ and an orientation $d \in D$, where $D = \{\perp, \vdash\}$, and outputs an element of $\mathbb{Z}^{0+}$. The orientation d is the relative position to each other with $\perp$ meaning vertical and $\vdash$ meaning horizontal, with the σ_1 being the west or north state respectively. A *transition rule* consists of two ordered pairs of states $(\sigma_1, \sigma_2), (\sigma_3, \sigma_4)$ and an orientation $d \in D$, where $D = \{\perp, \vdash\}$. This denotes that if the states (σ_1, σ_2) are next to each other in orientation d (σ_1 as the west/north state) they may be replaced by the states (σ_3, σ_4). An *assembly* A is a set of tiles with states in Σ such that for

*This research was supported in part by National Science Foundation Grant CCF-2329918.

†Department of Computer Science, University of Texas Rio Grande Valley

every pair of tiles $t_1 = (\sigma_1, p_1), t_2 = (\sigma_2, p_2), p1 \neq p2$. Informally, each position contains at most one tile.

Let $B_G(A)$ be the bond graph formed by taking a node for each tile in A and adding an edge between neighboring tiles $t_1 = (\sigma_1, p_1)$ and $t_2 = (\sigma_2, p_2)$ with a weight equal to $\Pi(\sigma_1, \sigma_2)$. We say an assembly A is $\tau-$stable for some $\tau \in \mathbb{Z}^0$ if the minimum cut through $B_G(A)$ is greater than or equal to τ.

A *Seeded Tile Automata* system is a $6-$tuple $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, s, \tau)$ where Σ is a set of states, $\Lambda \subseteq \Sigma$ a set of initial states, Π is an affinity function, Δ is a set of transition rules, s is a stable assembly called the seed assembly, and τ is the temperature (or threshold). A tile $t = (\sigma, p)$ may attach to an assembly A at temperature τ to build an assembly $A' = A \cup t$ if A' is $\tau-$stable and $\sigma \in \Lambda$. We denote this as $A \rightarrow_{\Lambda,\tau} A'$. An assembly A can transition to an assembly A' if there exist two neighboring tiles $t_1 = (\sigma_1, p_1), t_2 = (\sigma_2, p_2) \in A$ (where t_1 is the west or north tile) such that there exists a transition rule in Δ with the first pair being (σ_1, σ_2), the second pair being some pair of states (σ_3, σ_4) such that $A' = (A \setminus \{t_1, t_2\}) \cup \{t_3 = (\sigma_3, p_1), t_4 = (\sigma_4, p_2)\}$. We denote this as $A \rightarrow_\Delta A'$. For this paper, we focus on systems of temperature $\tau = 1$, and all bond strengths are equal to 0 or 1.

An assembly sequence $\overrightarrow{\alpha} = \{\alpha_0, \alpha_1, \ldots\}$ in Γ is a (finite or infinite) sequence of assemblies such that each $\alpha_i \rightarrow_{\Lambda,\tau} \alpha_{i+1}$ or $\alpha_i \rightarrow_\Delta \alpha_{i+1}$. An assembly subsequence $\beta = \{\alpha'_0, \alpha'_1, \ldots\}$ in Γ is a (finite or infinite) sequence of assemblies such that for each α'_i, α'_{i+1} there exists an assembly sequence $\overrightarrow{\alpha} = \{\alpha'_i, \ldots, \alpha'_{i+1}\}$.

We define the *shape* of an assembly A, denoted $(A)_\Lambda$, as the set of points $(A)_\Lambda = \{p | (\sigma, p) \in A\}$.

Discrete Self-Similar Fractals. Let $1 < c, d \in \mathbb{N}$ and $X \subseteq \mathbb{N}^2$. We say that X is a $(c \times d)$-discrete self-similar fractal if there is a set $G \subseteq \{0, \ldots, c-1\} \times \{0, \ldots, d-1\}$ with $(0,0) \in G$, such that $X = \bigcup_{i=1}^{\infty} X_i$, where X_i is the i^{th} *stage* of G satisfying $X_0 = \{(0,0)\}$, $X_1 = G$, and $X_{i+1} = \{(a,b) + (c^i v, d^i u) | (a,b) \in X_i, (v,u) \in G\}$. In this case, we say that G *generates* X. We say that X is a discrete self-similar fractal if it is a $(c \times d)$-discrete self-similar fractal for some $c, d \in \mathbb{N}$. A generator G is termed *feasible* if it is a connected set, and there exist (not necessarily distinct) points $(0,y), (c-1,y), (x,0), (x,d-1) \in G$, i.e., a pair of points on each opposing edge of the generator bounding box that share the same row or column. Note that the fractal generated by a generator is connected if and only if the generator is feasible. For the remainder of this paper we only consider feasible generators.

Strict and Super-strict. Let X be a discrete self-similar fractal with feasible generator G. Consider a seeded TA system $\Gamma = (\Sigma, \Lambda, \Pi, \Delta, s, \tau)$ with $(s)_\Lambda = G$, and let S denote the set of all valid assembly sequences for Γ. Γ *strictly* builds X if $\forall \overrightarrow{\alpha_i} = \{s, \alpha_1, \ldots, \alpha_i, \ldots\} \in S$, $\overrightarrow{\alpha_i}$ is infinite and $\lim_{i\to\infty}(\alpha_i)_\Lambda = X$. We further say that Γ *super-strictly* builds discrete self-similar fractal X if $\forall \overrightarrow{\alpha_i} \in S$, there exists a subsequence $\beta = \{s, \alpha'_1, \ldots\}$ of $\overrightarrow{\alpha_i}$ such that each $(\alpha'_i)_\Lambda = X_i$.

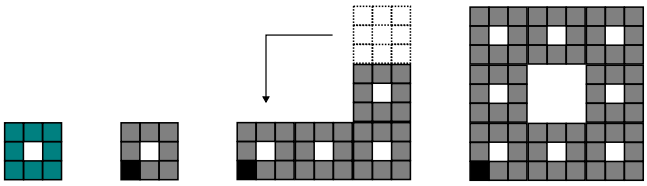


Figure 1: From left to right: the generator, the seed assembly (with the tile in black representing the origin tile), the assembly at the start of step 4 and the assembly at stage 2 (or the end of stage 1).

Other Definitions. Let G be a feasible generator with corresponding points $(0,y), (c-1,y), (x,0), (x,d-1) \in G$, X be the discrete self-similar fractal corresponding to G, and A be an assembly such that $(A)_\Lambda = X_i$ for some $i \in \{1, 2, \ldots\}$. We denote *key positions* as four points $p_N, p_E, p_W, p_S \in (A)_\Lambda$ satisfying $p_N = (x + c^{i-1} \cdot x, d^i - 1)$, $p_E = (c^i - 1, y + d^{i-1} \cdot y)$, $p_W = (0, y + y \cdot d^{i-1})$ and $p_S = (x + c^{i-1} \cdot x, 0)$. The four tiles $t_N, t_E, t_W, t_S \in A$ with positions p_N, p_E, p_W, p_S, respectively, are called *key tiles*. We denote $t_0 \in G$ the *origin tile* if t_0 has position $(0,0)$.

Let $G_G = (V, E)$ be the embedded graph formed by adding a vertex for each point $p \in G$ and adding an edge between vertices representing neighboring points $p_1, p_2 \in G$. Let $H = \langle h_0, \ldots, h_m \rangle$ $(m = |G| - 1)$ denote a Hamiltonian path in G_G, and let vertex h_0 represent the origin of the generator, where each h_j represents $p_j = (w_j, u_j)$. Given X_i, the i^{th} stage of generator G, denote $X_i^j = \{(a + c^i w_j, b + d^i u_j) \mid (a,b) \in X_i, j \in \{0, \ldots, m\}\}$, where j is the j^{th} *step* for stage i.

Additionally, we denote a particular assembly A as A_i if $(A)_\Lambda = X_i$ and A as A_i^J if $(A)_\Lambda = \bigcup_{k=0}^{J} X_i^k$, where $J \in \{0, \ldots, m\}$. To refer to a specific sub-assembly of A_i^J corresponding to step $j \in \{0, \ldots, J\}$ for stage i, we use A_i^j, where $A_i^j \subset A_i^J$ and $(A_i^j)_\Lambda = X_i^j$.

3 Construction

Given a feasible generator G where the resulting embedded graph G_G has a Hamiltonian path, we construct a seeded TA system with seed s $((s)_\Lambda = G)$ and origin tile t_0 that super-strictly builds the corresponding fractal infinitely. To start, the number of points m in the generator excluding the origin determines the number of steps m needs to scale the assembly from stage i to stage $i+1$. We denote each point in the generator as p_j, where j represents the distance from itself and the origin p_0 following a selected Hamiltonian path P in G_G.

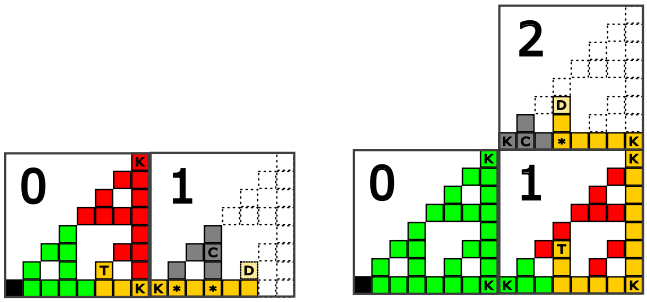


Figure 2: The Sierpinski triangle starting from stage 3 with $m = 2$ steps. The tile marked T is sending a signal (yellow) to place itself at position D. Tiles marked K are key tiles (origin tile is also a key tile, just not labeled). Tiles marked $*$ have a cap in the direction of the gray placed tiles. Tiles marked C had 2 caps, so the cap shifted to the tiles marked $*$.

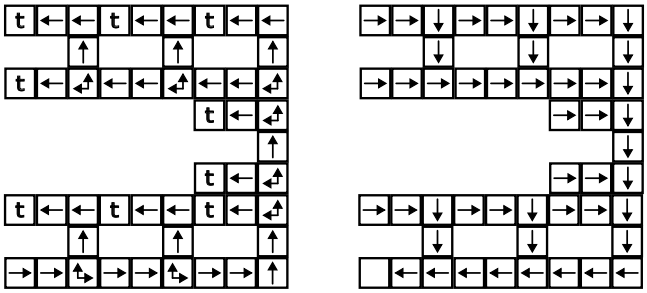


Figure 3: An example of the 'next' (left) and 'previous' (right) directions for each tile. Tiles marked t are terminal tiles, meaning they have no 'next' direction. The only tile without a 'previous' direction is the origin tile, which is the tile located at the bottom left. Note that the 'next' and 'previous' directions at each tile do not always include all adjacent tiles.

If we let $d(p_j, p_{j-1})$ denote the relative position of p_j to p_{j-1} (north, east, west or south), then we can represent the sequence of directions the assembly will grow.

The high-level idea of the construction is as follows: given an initial assembly A^0, translate a copy of A^0 in the direction of $d(p_1, p_0)$ and denote the copy as A^1. Repeat for all A^j, copying A^j in direction $d(p_{j+1}, p_j)$ until $j = m$. Set $A' = \bigcup_{j=0}^{m} A^j$, and then continue the process with $A^0 = A'$. See Figure 1 for an example.

However, since the seeded TA model is limited to single attachments and transitions, a direct implementation of this high-level idea is not possible. Instead, we give each tile the responsibility of placing itself in the correct location, with the final result being a copied translation from A^j to A^{j+1}. Thus, a crucial part of our construction is the ability to store information and send signals through the assembly. This section focuses on describing each of these components more thoroughly.

3.1 Storing Information

In order to correctly copy the base assembly, every tile needs specific information. This information can implicitly be stored as the state σ of the tile.

Current State ($STATE(t)$)**.** As each tile is responsible for placing itself in the right location at the next step, it is important to know whether each tile has either 1) not placed itself yet, 2) is currently placing itself or 3) has already placed itself. $STATE(t)$ denotes the current state of tile t. In Figure 2, green tiles are tiles with $STATE(t) =$ complete, red tiles are tiles with $STATE(t) =$ incomplete and the yellow tile marked T has $STATE(t) =$ waiting. Gray tiles are tiles that have been placed from the current step, so they must wait until the current step finishes.

Direction to Key Tiles ($KEY_d(t)$)**.** With the key tile information, signals are sent in the direction of the correct key tile. For instance, if the assembly is being copied to the north at step j, signals are sent in the direction of $t_N^j \in A_i^j$. $KEY_d(t)$ denotes this direction for t. To reference all 4 key tiles, we use $KEY_{NEWS}(t)$. This is illustrated in Figure 5a.

Next/Previous Tiles ($NEXT(t)/PREV(t)$)**.** This serves the purpose of knowing where each tile's neighbors are (or should be). $NEXT(t)$ denotes the direction to the 'next' tiles from t, which usually signifies which directions the signal can propagate, excluding the source direction. $PREV(t)$ denotes the direction to the previous tile from t, which usually signifies the direction a signal comes from. We use $NEXT_t(t)$ to denote the tile adjacent to t in direction $NEXT(t)$ (or similarly, the set of tiles adjacent to t for each direction in $NEXT(t)$). Similarly, $PREV_t(t)$ denotes the tile adjacent to t in direction $PREV(t)$. This is described in Figure 3.

The State of Neighboring Sub-assemblies ($SUB_d(t)$)**.** This is crucial for several reasons. Firstly, this creates the order in which tiles are placed. Secondly, this makes it possible to keep track of which direction the signal is coming from, and once the tile is placed, where the signal needs to return to. $SUB_d(t)$ denotes the state of the sub-assembly (whether all tiles have been placed or not) stemming from the neighboring tile of t in direction d. To reference the state of all sub-assemblies adjacent to t, we use $SUB_{NEWS}(t)$. Additionally, we refer to a specific sub-assembly as $SUBASM_d(t)$, denoting the sub-assembly stemming from tile t in direction d. See Figure 5b for an example.

The Tile being Transferred ($TRANS(t)$)**.** To distinguish between different signals, each tile keeps track of which tile the signal started from. $TRANS(t)$ is used to denote the tile that the signal is coming from.

Step. Each tile stores which step it is a part of. This allows an assembly A_i^J to know which tiles to use (tiles in A_i^j) to create sub-assembly A_i^{j+1}. Additionally, A_i^j will only send signals to $t^j_{d(p_{j+1},p_j)}$.

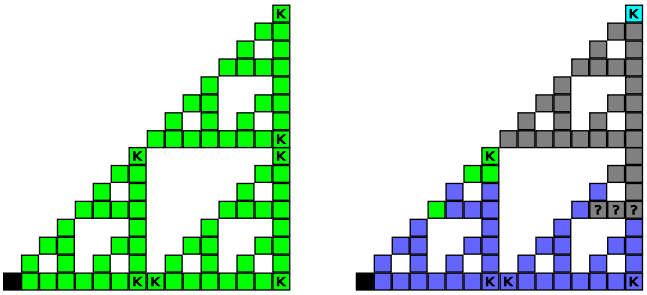


Figure 4: The Sierpinski triangle resetting at the end of stage 3. Tiles marked ? are waiting for the sub-assemblies adjacent to reset (the blue tiles to the north and west). Gray tiles have been reset. Blue tiles are transmitting the reset signal. Light blue tiles marked 'K' are the new key tiles for the assembly.

Terminal ($TERM(t)$)**.** Tiles must know when they are at the end of a sub-assembly. Once a terminal tile is placed, the system knows part of the sub-assembly is complete. $TERM(t)$ is a boolean that denotes whether tile t is terminal or not. In Figure 3, these tiles are marked t.

Caps ($CAP_d(t)$)**.** Copied assemblies require one extra piece of information. As the shape grows, there are points where signals can branch in multiple directions. To direct this, signals always go 'left' when there is a fork. If the sub-assembly in this direction is already constructed, a cap is placed to prevent signals from going in that direction, and it instead goes to the next path. If all paths have a cap, then it turns around and the cap is shifted to reflect that all paths are complete. Caps start from terminal tiles and gradually shift as the sub-assemblies are completed. $CAP_d(t)$ is a boolean that denotes whether tile t has a cap in direction d. Figure 2 shows an example of how caps are used and shifted.

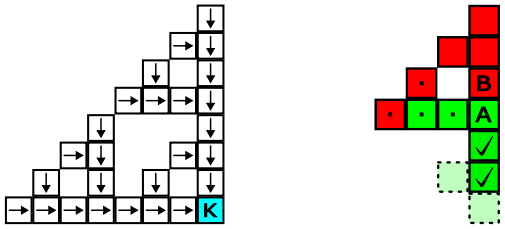
3.2 Signal Passing

In addition to the stored information, it is important that tiles can communicate through signal passing. This is done via transition rules.

Tile Placement Ordering. The order in which tiles place themselves follows a 'left' first order (described in Section 3.3). As the tiles place themselves and are marked complete, transition rules prompt the next tile to start placing itself.

Tile Placement Signals. When a tile t_i^j is placing itself, the signal is transmitted from t_i^j to the tile adjacent to the target position for t_i^j. Transition rules make this possible by transferring the signal between adjacent tiles. In Figure 2, the transmission of this placement signal is represented as the sequence of yellow tiles.

Tile Placement Completion Signals. Once the tile is placed in the correct location, a 'completion' signal gets sent back the same direction as the placement signal. Once this signal reaches the tile getting placed from A_i^j, the tile is marked as complete.



(a) (b)

Figure 5: (a) An example of the direction stored at each tile for t_E, the tile marked K. (b) An example of how sub-assemblies work. The check mark denotes the sub-assemblies to the south of tile A are completed. The sub-assemblies to the west and north of A, however, are not. As a result, $SUB_S(A)$ = complete, $SUB_W(A)$ = incomplete and $SUB_N(A)$ = incomplete. Tiles marked with . are part of $SUBASM_W(A)$. Note that B does not start placing itself until $SUB_W(A)$ and $SUB_S(A)$ are both marked completed.

Cap Signals. When a terminal tile is placed, as the 'completion' signal gets sent back in the sub-assembly being created, a 'cap' is sent back with it to mark the sub-assembly as complete. This forces future signals to go a different path to complete a different sub-assembly.

Reset Signals When a stage is completed, reset signals are sent to update current state, direction to key tiles, state of neighboring tiles and step, as well as removing any remaining caps. Figure 4 illustrates the resetting process.

Figure 2 details the construction outlined in Sections 3.1 and 3.2. The following section provides more specific details to express how the system interacts to create these fractals.

3.3 Approach

This section describes the process for taking an assembly A_i to A_{i+1}, assuming G is a feasible generator for $(A_i)_\Lambda$, $H = \langle h_0, \ldots, h_m \rangle$ is a Hamiltonian path in G_G starting from the origin where m is the number of steps, i is the stage and $t_0 \in A_i^0$ is the origin tile. Additionally, we denote t_d^j as the key tile for direction d in assembly A_i^j and $d_j = d(p_{j+1}, p_j)$. We briefly define some additional terminology:

$OPP(d)$**.** This denotes the complement of direction d, e.g., $OPP(north) = south$.

$LEFT(D)$**.** Consider $D \in \{\{N\}, \{E\}, \{W\}, \{S\}, \{N, E\}, \{E, S\}, \{N, W\}, \{W, S\}\}$, where N, E, W, S represent north, east, west and south respectively. $LEFT(D)$ denotes the 'left' direction for D. This is 1) North if $D = \{N, E\}$ or $\{N\}$, 2) East if $D = \{E, S\}$ or $\{E\}$, 3) West if $D = \{N, W\}$ or $\{W\}$ and 4) South if $D = \{W, S\}$ or $\{S\}$.

Conversely, $RIGHT(D)$ denotes $NEXT(D) \setminus LEFT(D)$. For a tile t, we use $LEFT_t(D)/RIGHT_t(D)$

to denote the tile adjacent to t in direction $LEFT(D)/RIGHT(D)$, respectively.

RESET(t)**.** At the end of stage i, tiles must reset. This includes 1) updating the direction to the new 4 key tiles and 2) updating the 'next' direction for former key tiles (see Figure 6). $RESET(t)$ denotes tile t resetting, defined as follows:

- If $t = t^j_{d_j}$, where $p_j \in X_1$ is the key position for d_j, set $KEY_{d_j}(t) =$ current tile.
- For each tile $t_a \in NEXT_t(t)$ (if all t_a have reset) set each $KEY_d(t) = PREV(t)$ if $KEY_d(t_a) = d(t, t_a)$. If there is a tile such that $KEY_d(t_a) \neq d(t, t_a)$, set $KEY_d(t) = KEY_d(t_a)$. If t_a is a key tile for d, set $KEY_d(t) = d(t_a, t)$.
- Clear $TERM(t_c)$ and update $NEXT(t_c)$ if appropriate.

3.3.1 Algorithm

Now we describe the algorithm. For better comprehension, we describe the algorithm using sub-processes. Technical descriptions of these sub-processes are included in Section 6.

Start with $j = 0$ and let $t_c = t_0$ denote the current tile getting placed, starting with the origin tile. Let A_i^{j+1} represent the translated assembly being created at step $j + 1$. The following will be repeated until $j = m$. While $SUB_{PREV(t^j_{d_j})}(t^j_{d_j})$ and $SUB_{NEXT(t^j_{d_j})}(t^j_{d_j})$ are not marked as completed:

1. Let t_a denote the tile adjacent to t_c in direction $KEY_{d_j}(t_c)$. Run *send_placement_signal*$(t_c, t^j_{OPP(d_j)}, t^j_{d_j}, t_a, d_j)$ to send a signal through the assembly to place t_c in the correct location.
2. The placement signal stops at the tile adjacent to the target position by always traversing 'left' until a tile no longer exists. Run *place_tile*(t_a, t'_c, p) to place the tile at this location, where t_a is the tile adjacent to position p, the target position for t_c. This places t_c in the correct location as t'_c.
3. Retrace the signal to the tile that got placed by running *send_completion_signal*(t'_c, t_a, d_j). This also marks the tile as complete.
4. Mark sub-assemblies as complete if needed. Run *mark_completed_sub-assemblies*(t_c, t_a) if $TERM(t_c) =$ True, where t_c is the tile that just placed itself and t_a is the tile adjacent to t_c such that $STATE(t_a) =$ complete.
5. Choose the next tile to be placed. Let t_c denote the last tile updated and C be the tiles in $NEXT_{t_c}(t_c) \cup PREV_{t_c}(t_c)$ that have a completed state. If $t_c \neq t^j_{d_j}$, repeat from (1) with the new $t_c = LEFT_{t_c}(PREV(t_c) \cup NEXT(t_c) \setminus C)$.

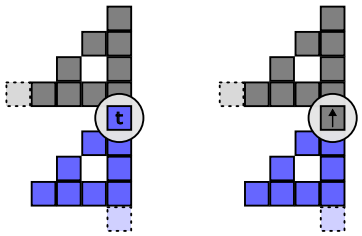


Figure 6: The highlighted tile is initially set as terminal. Since the tile used to be a key tile, resetting also updates the 'next' direction if appropriate.

6. If $t_c = t^j_{d_j}$, $j \neq m$ (the stage is not yet completed) and $SUB_{PREV(t^j_{d_j})}(t^j_{d_j})$ and $SUB_{NEXT(t^j_{d_j})}(t^j_{d_j})$ are marked as completed (every tile has now been placed), run *start_next_step*$(t^{j+1}_{OPP(d_j)}, t^{j+1}_{d_{j+1}}, d_j, d_{j+1})$ to signal for the next sub-assembly to start being created. Repeat from (1) with $j = j + 1$, $A_i^j = A_i^{j+1}$ and clear $TRANS(t_c)$.
7. If instead $j = m$, the initial assembly has now been up-scaled and has reached the end of stage i. To repeat this process, the assembly now has to reset. Run *reset*$(t^m_{OPP(d_{m-1})})$.
8. Repeat the algorithm.

A primary reason as to why this algorithm works is the existence of a Hamiltonian path in the generator, as this dictates the directions in which the fractal grows. This allows growth of the fractal for any step to only depend on the created sub-assembly from the previous step, regardless of whether or not the resulting assembly contains a Hamiltonian path or not. If a generator does not contain a Hamiltonian path, then some sub-assemblies of the fractal must be used multiple times to create copies in multiple directions, which results in synchronicity issues as multiple signals could exist in the assembly at once.

4 Results

We now show that any feasible generator G with a Hamiltonian path in G_G can be super-strictly built by a seeded TA system Γ. Let G be a feasible generator for discrete self-similar fractal X, with $H = \langle h_0, \ldots, h_m \rangle$ denoting a Hamiltonian path in G_G such that each h_j corresponds to point $p_j = (w_j, u_j) \in G$. Let $d_j = d(p_{j+1}, p_j)$, A be the current assembly starting from $A = A_i^J$ for some $J \in \{0, \ldots, m-1\}$, and $A_i^{j+1'} = A \setminus A_i^J$. We denote the copy of a tile t as t'.

Lemma 1 *Under the construction from Section 3, tile $t^j_{OPP(d_j)} \in A_i^j$ must be the first tile to place itself.*

Proof. This is due to geometry. While there may be other adjacent tiles between A_i^j and the sub-assembly

being created, $A_i^{j+1'}$, $t_{d_j}^j$ is the only tile that recognizes the existence of $A_i^{j+1'}$. Thus, the only way to send a signal to $A_i^{j+1'}$ is through $t_{d_j}^j$, and the only adjacent tile to $t_{d_j}^j$ in $A_i^{j+1'}$ is $t_{OPP(d_j)}^{j'}$. □

Lemma 2 *Let $t_i = (\sigma_i, p_i) \in A_i^j$ and let $t_f = (\sigma_f, p_f) \in A_i^{j+1'}$ represent the tile adjacent to p_i', the target location for t_i'. A signal will follow exactly 1 path from t_i to t_f.*

Proof. From Section 3.3, signals will always follow one path in A_i^j and $A_i^{j+1'}$. We show that this signal ends at tile t_f. This can be done by comparing the order in which tiles are chosen to be placed to the direction that the signal travels.

To prove equivalence, we show that for each tile in $A_i^{j+1'}$, $PREV(t_c) \cup NEXT(t_c) \setminus d(t_p, t_c) = NEXT(t_c^*)$. We consider 2 cases:

Case 1: $t_c^* = t_{OPP(d_j)}^{j+1}$. From Section 3.3, $NEXT(t_c^*) = NEXT(t_{OPP(d_j)}^j) \cup PREV(t_{OPP(d_j)}^j) \setminus OPP(d_j)$. Initially, t_p is the tile in direction $OPP(d_j)$ from t_c. This results in $PREV(t_c) \cup NEXT(t_c) \setminus d(t_p, t_c)$.

Case 2: t_c^* is any other tile. Let t_p^* denote the tile adjacent to t_c^* from which the signal came from, with t_p, t_c denoting the corresponding tiles from A_i^j. We consider 2 scenarios.

1. $t_c \in NEXT_{t_p}(t_p)$. From Section 3.3, $NEXT(t_c^*) = PREV(t_c) \cup NEXT(t_c) \setminus d(t_p, t_c)$.
2. $t_c \in PREV_{t_p}(t_p)$. From Section 3.3, $NEXT(t_c^*) = NEXT(t_c) \cup PREV(t_c) \setminus (d(t_p, t_c) \cup \{\textit{directions to tiles not in step } j\})$. The only time there exists a direction to a tile not in step j is when t_c is the first tile placed in step j. Since this is no longer the case for step $j+1$, we get rid of this direction from $NEXT(t_c^*)$, and since the next tile chosen to be placed from t_c does not consider this direction either, the equivalence holds. □

Lemma 3 *Let $t_i = (\sigma_i, p_i) \in A_i^j$ and let $t_f = (\sigma_f, p_f) \in A_i^{j+1'}$ represent the tile adjacent to p_i', the target location for t_i'. Tile t_f will place tile t_i' at position $p_i' = p_i + a \cdot d_j^*$, where $a - 1 \in \mathbb{N}$ represents the distance $|p_{d_j} - p_{OPP(d_j)}|$ and $d_j^* \in \{0, 1\}^2$ is a 2-D vector denoting the direction.*

Proof. Let $d_j^* = [0, 1], [1, 0], [-1, 0], [0, -1]$ represent d_j = north, east, west and south, respectively. In the case of $t_{OPP(d_j)}$' being the tile placed, the signal will stop at tile $t_{d_j}^j$ with position $p_{d_j} = p_{OPP(d_j)} + (a-1) \cdot d_j^*$. Tile $t'_{OPP(d_j)}$ is then placed at position $p'_{OPP(d_j)} = p_{d_j} + d_j^* = p_{OPP(d_j)} + a \cdot d_j^*$.

For any other tile t_i, as described in Lemma 2, we know 2 things: 1), the signal from tile t_i will stop at tile t_f adjacent to position p_i' and 2) the order in which tiles are chosen to be placed is equivalent to the direction in which signals are passed. Since the relative position of p_i' to $t'_{OPP(d_j)}$ is the same as p_i to $t_{OPP(d_j)}$ and $t'_{OPP(d_j)} = t_{OPP(d_j)} + a \cdot d_j$, it follows that $p_i' = p_i + a \cdot d_j^*$. □

Lemma 4 *Let $t_i = (\sigma_i, p_i) \in A_i^j$ and let $t_i' = (\sigma_i', p_i') \in A_i^{j+1'}$ represent the copy of t_i. A signal will follow exactly 1 path from t_i' to t_i.*

Proof. By Lemma 2, there exists one path from p_i to the tile adjacent to p_i'. Thus, when tile t_i' is placed at position p_i', the converse holds true by retracing this path. □

Theorem 5 *There is at most one tile transmitting a signal in A.*

Proof. By contradiction. Assume that there exists 2 tiles $t_1 \neq t_2$ transmitting signals through A_i^j and let $A^{j_1^*}, A^{j_2^*} \subset A_i^j$ denote 2 sets of tiles such that:

1. $\forall t_a \in A^{j_1^*}$, $STATE(t_a) =$ complete.
2. $\forall t_b \in NEXT_{t_a}(t_a) \cup PREV_{t_a}(t_a)$, $t_b \in A^{j_1^*}$, $STATE(t_b) =$ incomplete or $t_b = t_1$.

where the same applies for $A^{j_2^*}$ and t_2. We consider 2 cases:

Case 1: $A^{j_1^*} \cap A^{j_2^*} = \emptyset$. This implies $t_{OPP(d_j)}^j \in A^{j_1^*}$ or $t_{OPP(d_j)}^j \in A^{j_2^*}$, but not both. By Lemma 1, $t_{OPP(d_j)}^j$ must be the first tile placed, resulting in a contradiction.

Case 2: $A^{j_1^*} \cap A^{j_2^*} \neq \emptyset$. Consider a tile t^* such that $t_1 \in SUBASM_{d_1}(t^*)$ and $t_2 \in SUBASM_{d_2}(t^*)$. Since $SUBASM_{RIGHT(\{d_1,d_2\})}(t^*)$ must wait for $SUBASM_{LEFT(\{d_1,d_2\})}(t^*)$ to be completed, it must be that $d_1 = d_2$. This implies that $t_1 = t_2$. □

Theorem 6 *Step $j+2$ will start only when step $j+1$ is completed.*

Proof. By our construction, since $t_{d_j}^j$ is the tile communicating between A_i^j and $A_i^{j+1'}$, both $SUB_{PREV(t_{d_j})}(t_{d_j})$ and $SUB_{NEXT(t_{d_j})}(t_{d_j})$ must be marked as completed before step $j+2$ begins. This is true only when $\forall t \in A_i^j$, $STATE(t) =$ complete. □

Lemma 7 *Let $A_i^M = \bigcup_{j=0}^{m} A_i^j$ denote the resulting assembly at step m for stage i. Every tile will reset before moving to stage $i+1$.*

Proof. This is due to our construction. A tile t will only reset when $\forall t_a \in NEXT_t(t)$, t_a is reset. The only time this is not true is when t is terminal, which marks the end of a sub-assembly. □

Lemma 8 *Let $A_i^M = \bigcup_{j=0}^{m} A_i^j$ denote the resulting assembly at step m for stage i. When t_0 resets, there will exist at most 4 key tiles and $KEY_{NEWS}(t) \forall t \in A_i^M$ is updated to point to these new key tiles.*

Proof. We first show that there will exist at most four key tiles, one for each direction d. From Section 3.3, t_d must appear only in some step j. Thus, as the assembly resets, t_d^j resets as the new t_d for the up-scaled assembly, and all other $t_d^k \; \forall k \neq j \in \{0, \ldots, m\}$ are reset to normal tiles. This leaves at most four key tiles.

Next, we show that every tile will point to the direction of the new t_d's. We show this by contradiction. Assume that there exists a tile t such that $KEY_d(t) = PREV(t)$, but $t_d \in SUBASM_{NEXT(t)}(t)$. This implies that for $t_a = PREV_{t_d}(t_d)$, $KEY_d(t_a) = PREV(t_a)$, which is true only if t_d is not the key tile for direction d. Hence, $t_d \notin SUBASM_{NEXT(t)}(t)$. □

Lemma 9 *Let $A^{j*} \subseteq A_i^j$ where $\forall t \in A^{j*}$, $STATE(t) =$ complete. At the end of step $j+1$, $(A_i^{j+1'})_\Lambda = (A_i^j)_\Lambda + a \cdot d_j^*$, where $a - 1 \in \mathbb{N}$ represents the distance $|p_{d_j} - p_{OPP(d_j)}|$ and $d_j^* \in \{-1, 0, 1\}^2$ is a 2-D vector denoting the direction.*

Proof. We use Lemmas 1 and 3 to construct an inductive proof. Let $d_j^* = [0, 1], [1, 0], [-1, 0], [0, -1]$ represent $d_j =$ north, east, west and south, respectively.

Base case. $|A_i^{j*}| = 0$, with $t_1 = t_{OPP(d_j)}^j$ being the first tile copying itself (Lemma 1). By Lemma 3, $t'_{OPP(d_j)}$ is placed at position $p'_1 = p_1 + a \cdot d_j^*$.

Inductive step. $|A_i^{j*}| = k$, with t_{k+1} being the tile copying itself. By Lemma 3, t'_{k+1} is placed at position $p'_{k+1} = p_{k+1} + a \cdot d_j^*$. It holds that $(A_i^{j+1'})_\Lambda = (A^{j*})_\Lambda + a \cdot d_j^*$. Thus, $(A_i^{j+1})_\Lambda = (A_i^j)_\Lambda + a \cdot d_j^*$. □

Theorem 10 *At the end of stage i, $(A_i^M)_\Lambda = (A_{i+1})_\Lambda$.*

Proof. Follows from Lemma 9. For each A_i^K with $K \in \{0, \ldots, m-1\}$, a new sub-assembly A_i^{k+1} is constructed such that the new assembly $A_i^{K+1} = A_i^K \cup A_i^{k+1}$ satisfying $(A_i^{K+1})_\Lambda = X_i^{K+1}$. Thus, the final assembly $A_i^M = A_i^{M-1} \cup A_i^m$ with $(A_i^M)_\Lambda = \bigcup_{j=0}^{m} X_i^j = X_{i+1} = (A_{i+1})_\Lambda$. □

Theorem 11 *Let X be a discrete self-similar fractal with feasible generator G in bounding box $c \times d$ such that G_G has a Hamiltonian path $\langle h_0, \ldots, h_m \rangle$ where h_0 represents the origin. There exists a seeded TA system Γ with $O(|G|)$ states, $O(|G|^2)$ transitions and $O(|G|^2)$ affinities that super-strictly builds X.*

Proof. We start by showing Γ strictly builds X. This follows from Theorem 10. We start with seed s, where $(s)_\Lambda = G$ and each $t_j \in s$ stores $NEXT(t_j) = d(p_{j+1}, p_j)$ (if p_{j+1} exists), $PREV(t_j) = d(p_{j-1}, p_j)$ (if p_{j-1} exists) and $TERM(t_m) =$ True. Denote the assembly as A_1. By Theorem 10, applying the construction from Section 3 yields a new assembly $A_2 = \bigcup_{j=0}^{m} A_i^j$ with shape $(A_2)_\Lambda = X_2$. Repeating this for all A_i yields $\lim_{i \to \infty} (A_i)_\Lambda = X$.

Now we show that Γ super-strictly builds X. To do so, we consider 2 cases:

1. The assembly A_i^M at the end of stage i before resetting. Leading up to this point, the order in which tiles are placed and signals are passed is deterministic. As a result, there exists 1 unique valid assembly sequence from A_i to A_i^M.
2. The assembly A_{i+1} after A_i^M resets. While there no longer exists 1 unique valid assembly sequence from A_i^M to A_{i+1}, Lemmas 7 and 8 show that every tile will reset to point to the 4 new key tiles. From (1), the rest of the local information at each tile will remain the same. Thus, every valid assembly sequence from A_i^M to A_{i+1} starts with A_i^M and ends with A_{i+1}.

Choose $\beta = \{s, A_2^M, A_3^M, \ldots\}$ or $\beta = \{s, A_2, A_3, \ldots\}$. It follows that Γ super-strictly builds X.

Disregarding steps, the total number of ways information can be locally stored at any tile is $O(1)$ since a tile has at most 4 neighbors. However, as tiles need to distinguish between different sub-assemblies representing different steps, this results in $O(|G|)$ different states. Similarly, since transition rules and affinities use combinations of 2 states, this results in $O(|G|^2)$ transition rules and affinity rules. □

5 Conclusion

In this paper, we present a method to strictly build fractals infinitely under the assumption that the generator is feasible and contains a Hamiltonian path. This contrasts with previously known results from similar (but slightly differing) models such as the aTAM, where some fractals, such as the Sierpinski triangle, are shown to be impossible to build strictly. Additionally, we show that this class of fractals can be super-strictly built, as our construction guarantees stopping at unique intermediate assemblies for all possible assembly sequences, where each intermediate assembly represents a different stage of the fractal. However, there remains several open questions:

- Our construction strictly builds fractals infinitely with states linear in the size of the generator and transitions and affinities quadratic in the size of the generator. Is there an alternative method that reduces the state, transition and affinity counts?

- Is it possible to construct all fractals infinitely? If not, what fractals are impossible to build?
- Does there exist a seeded TA system that can strictly build any fractal infinitely?
- Our work focuses on systems with temperature 1. Is it possible to take advantage of systems with higher temperatures to strictly build these fractals more efficiently, or does higher temperatures increase the complexity of the problem?

6 Full Details for Algorithm

Below are the full details for the sub-processes used in the algorithm described in Section 3.3.

send_placement_signal(t_c, $t^j_{OPP(d_j)}$, $t^j_{d_j}$, t_a, d_j):

1. Set $STATE(t_c) =$ waiting.
   - If $t_c = t^j_{OPP(d_j)}$, set $NEXT(t'_c) = NEXT(t_c) \cup PREV(t_c) \backslash OPP(d_j)$, $PREV(t'_c) = OPP(d_j)$ and $TERM(t'_c) =$ False.
   - Else if $STATE(t_a) =$ complete and the number of tiles from step j in $NEXT_{t_c}(t_c) \cup PREV_{t_c}(t_c)$ is $= 1$, set $TERM(t'_c) =$ True and $PREV(t'_c) = d(t_a, t_c)$.
   - Else if $STATE(t_a) =$ complete and $t_c \in PREV_{t_a}(t_a)$, set $NEXT(t'_c) = NEXT(t_c) \cup PREV(t_c) \setminus (d(t_a, t_c) \cup \{$*directions to tiles not in step j*$\})$ and $PREV(t'_c) = d(t_a, t_c)$.
   - Else, set $NEXT(t'_c) = NEXT(t_c)$ and $PREV(t'_c) = PREV(t_c)$.
   - Set $KEY_{NEWS}(t'_c) = KEY_{NEWS}(t_c)$, $TERM(t'_c) = TERM(t_c)$ if $TERM(t'_c)$ is not defined yet, $SUB_{d(t_c,t_a)}(t_a) =$ waiting and $TRANS(t_a) = t'_c$.
   - Let $t_c = t_a$.
2. While $t_c \neq t^j_{d_j}$:
   - Set $SUB_{d(t_c,t_a)}(t_a) =$ waiting and $TRANS(t_a) = TRANS(t_c)$.
   - Let $t_c = t_a$.
3. If no tile exists adjacent to t_c in direction d_j, stop. Otherwise, set $SUB_{OPP(d_j)}(t_a) =$ waiting and $TRANS(t_a) = TRANS(t_c)$.
4. Repeat the following:
   (a) Let $t_a = LEFT_{t_c}(NEXT(t_c))$ if $!CAP_{LEFT(NEXT(t_c))}(t_c)$, else set $t_a = RIGHT_{t_c}(NEXT(t_c))$.
   (b) If t_a exists, set $SUB_{d(t_c,t_a)}(t_a) =$ waiting and $TRANS(t_a) = TRANS(t_c)$. Set $t_c = t_a$ and repeat from (a).
   (c) If t_a does not exist, stop.

place_tile(t_c, t_c', p):

1. Place t'_c in position p and set $SUB_{d(t_c,t'_c)}(t'_c) =$ maybe. If $TERM(t'_c)$, set $SUB_{d(t_c,t'_c)}(t'_c) =$ maybe with cap.

send_completion_signal(t_c, t_a, d_j):

1. Set $SUB_{d(t_a,t_c)}(t_c)$ to its original state, clearing $TRANS(t_c)$ and changing $SUB_d(t_a) =$ waiting to $SUB_d(t_a) =$ maybe for the direction d that the signal came from.
2. If $length(NEXT(t_c)) =$ number of caps on t_c, set $SUB_d(t_a) =$ maybe with cap and clear the cap from t_c. Otherwise, set $SUB_d(t_a) =$ maybe and leave the cap on t_c in direction $LEFT(t_c)$.
3. If $STATE(t_a) =$ waiting, set $SUB_{d(t_a,t_c)}(t_c)$ to its original state, clearing $TRANS(t_c)$ and changing $STATE(t_a) =$ waiting to $STATE(t_a) =$ complete. Otherwise, set $t_c = t_a$, let t_a be the tile adjacent to t_c from which the signal came from and repeat from (1).

mark_completed_sub-assemblies(t_c, t_a):

1. Repeat the following until t_a is not updated:
   (a) Set $SUB_{d(t_c,t_a)}(t_a) =$ complete.
   (b) If $length(SUB_{NEWS}(t_c) =$ complete$) = length(NEXT(t_c) \cup PREV(t_c))$, set $t_c = t_a$ and let t_a be the tile next to t_c with $STATE(t_a) =$ complete and $SUB_{d(t_c,t_a)} =$ incomplete.

start_next_step($t^{j+1}_{OPP(d_j)}$, $t^{j+1}_{d_{j+1}}$, d_j, d_{j+1}):

1. Let $t_c = t^{j+1}_{OPP(d_j)}$. Set $TRANS(t_c) =$ ready. While $t_c \neq t^{j+1}_{d_{j+1}}$:
   (a) Let t_a denotes the tile adjacent to t_c in direction $KEY_{d_{j+1}}(t_c)$. Set $TRANS(t_a) = TRANS(t_c)$ and clear $TRANS(t_c)$. Then let $t_c = t_a$.

reset($t^m_{OPP(d_{m-1})}$):

1. Set $TRANS(t^m_{OPP(d_{m-1})}) =$ reset. For all tiles t_a adjacent to $t_c = t^m_{OPP(d_{m-1})}$ in directions $d \in NEXT(t_c) \cup PREV(t_c)$, set $TRANS(t_a) = TRANS(t_c)$ and set $t_c = t_a$.
2. If $TERM(t_c)$, set $t_c = RESET(t_c)$. For the tile $t_a \in PREV_{t_c}(t_c)$, set $SUB_{d(t_c,t_a)}(t_a) =$ done. Add $d(t_c, t_a)$ to $NEXT(t_a)$ if not already done.
3. If $length(SUB_{NEWS}(t_c) =$ done $) = length(NEXT(t_c))$, then for the tile $t_a = PREV_{t_c}(t_c)$, set $t_c = RESET(t_c)$ and set $SUB_{d(t_c,t_a)}(t_a) =$ done.

References

[1] R. M. Alaniz, D. Caballero, S. C. Cirlos, T. Gomez, E. Grizzell, A. Rodriguez, R. Schweller, A. Tenorio, and T. Wylie. Building squares with optimal state complexity in restricted active self-assembly. *Journal of Computer and System Sciences*, 138:103462, 2023.

[2] A. A. Cantu, A. Luchsinger, R. Schweller, and T. Wylie. Signal Passing Self-Assembly Simulates Tile Automata. In Y. Cao, S.-W. Cheng, and M. Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[3] C. T. Chalk, D. A. Fernandez, A. Huerta, M. A. Maldonado, R. T. Schweller, and L. Sweet. Strict self-assembly of fractals using multiple hands. 76(1):195–224, sep 2016.

[4] J. Hendricks, M. Olsen, M. J. Patitz, T. A. Rogers, and H. Thomas. Hierarchical self-assembly of fractals with signal-passing tiles. *Natural computing*, 17:47–65, 11 2018.

[5] J. Hendricks and J. Opseth. Self-assembly of 4-sided fractals in the two-handed tile assembly model. In M. J. Patitz and M. Stannett, editors, *Unconventional Computation and Natural Computation*, pages 113–128, Cham, 2017. Springer International Publishing.

[6] J. Hendricks, J. Opseth, M. J. Patitz, and S. M. Summers. Hierarchical growth is necessary and (sometimes) sufficient to self-assemble discrete self-similar fractals. *Natural computing*, 13:357–374, 12 2020.

[7] J. E. Padilla, M. J. Patitz, R. T. Schweller, N. C. Seeman, S. M. Summers, and X. Zhong. Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. *International Journal of Foundations of Computer Science*, 25:459–488, 2014.

[8] M. J. Patitz. An introduction to tile-based self-assembly. In J. Durand-Lose and N. Jonoska, editors, *Unconventional Computation and Natural Computation*, pages 34–62, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] M. J. Patitz and S. M. Summers. Self-assembly of discrete self-similar fractals. *Natural computing*, 9:135–172, 08 2010.

[10] P. W. K. Rothemund. *Theory and experiments in algorithmic self -assembly*. PhD thesis, 2001.

[11] P. W. K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of dna sierpinski triangles. *PLOS Biology*, 2(12):null, 12 2004.

[12] E. Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, 1998.

List of Authors