Proceedings of the 35th Canadian Conference on Computational Geometry (CCCG 2023)
Montreal, Canada

Denis Pankratov, Concordia University

July 31 - August 4, 2023

Credits:
Logo design: Erik D. Demaine, Martin L. Demaine, Anna Lubiw. The CCCG 2001 Logo. Proceedings of CCCG 2001, pp. iv-v, 2001.

Welcome from Denis Pankratov, Conference Chair

This volume contains the papers presented at the

CCCG 2023: 35th Canadian Conference on Computational Geometry.

CCCG 2023 was co-located with the 18th Algorithms and Data Structures Symposium (WADS 2023). Both conferences took place at Concordia University, Montreal, Quebec, Canada. WADS was held on July 31 - August 2, 2023, followed by CCCG which was held on August 2 - August 4, 2023.

There were 49 submissions to CCCG 2023. One of the submissions was withdrawn during the review process. Almost all of the remaining 48 submissions (except for a couple) were reviewed by 3 program committee members. The committee decided to accept 36 papers. We thank the authors of all submitted papers and all those who have registered to attend the conference. We especially thank the invited speakers: Dr. Diane Souvaine (the Memorial Lecture for Paul Erdős) and Dr. Ileana Streinu (the Joint Memorial Lecture for Hurtado and Toussaint). In addition, we are grateful for the support and assistance provided by Concordia University and its members: in particular, the Department of Computer Science and Software Engineering, the Hospitality Services, and the graduate student volunteers.

We acknowledge the generous support from our sponsors: the Office of Vice-President, Research and Graduate Studies at Concordia University, the Gina Cody School of Engineering at Concordia University, the Fields Institute for Research in Mathematical Sciences, Elsevier, Wiley.

Sponsors

UNIVERSITÉ Concordia UNIVERSITY

UNIVERSITÉ Concordia UNIVERSITY

GINA CODY
SCHOOL OF ENGINEERING
AND COMPUTER SCIENCE

FIELDS

NON SOLUS

ELSEVIER

WILEY

Program Committee

Mohammad Ali Abam	Sharif University
Yeganeh Bahoo	Toronto Metropolitan University
Binay Bhattacharya	Simon Fraser University
Therese Biedl	University of Waterloo
Ahmad Biniaz	University of Windsor
Prosenjit Bose	Carleton University
Mirela Damian	Villanova University
Mark De Berg	Eindhoven University of Technology
Stephane Durocher	University of Manitoba
David Eppstein	University of California, Irvine
Hovhannes Harutyunyan	University of Concordia
Meng He	Dalhousie University
Shahin Kamali	York University
Akitoshi Kawamura	Kyoto University
Evangelos Kranakis	Carleton University
Daniel Krizanc	Wesleyan University
Yaqiao Li	Concordia University
Anil Maheshwari	Carleton University
Yoshio Okamoto	The University of Electro-Communications
Denis Pankratov (chair)	University of Concordia
Katharine Turner	Australian National University
Ryuhei Uehara	Japan Advanced Institute of Science and Technology

Invited Speakers

Diane Souvaine	Tufts University
Ileana Streinu	Smith College

Conference Program

Day 1 - August 2, 2023

Day 2 - August 3, 2023

Spanning Tree, Matching, and TSP for Moving Points: Complexity and Regret

Nathan Wachholz Subhash Suri*

Abstract

We explore the computational complexity, and regret, of some geometric structures under the recently introduced moving point model of Akitaya et al. [3]. Specifically, we want to build a single geometric structure (e.g. spanning tree, matching, or traveling salesman path) whose maximum cost during the motion of the input points is minimized. We call these structures, whose cost (sum of edge lengths) changes with the motion of points but whose topology remains fixed, *minimum moving point spanning tree* (MMST), *minimum moving point matching* (MMM), and *minimum moving point traveling salesman path* (MMTSP), respectively. We focus on linear motion of points in one dimension and prove the following results: (1) each of these problems is (weakly) NP-hard in one dimension, (2) remains NP-hard even under *radial expansion* where all points are moving away from a center, (3) remains NP-hard even if points are all moving with the *same speed.* A fixed topology is attractive in that it avoids expensive and continuous recomputation as input points move but it is inevitably sub-optimal. To quantify this tradeoff, we define the *regret* as the worst-case ratio between the cost of an optimal moving point structure and the maximum cost for the same input when the structure is continuously updated. We show the following results: (4) the regret ratio is $\Omega(\sqrt{n})$ for all three problems even in one dimension, but (5) has a tight bound of 2 for MMST and MMTSP if all points are moving with the same speed. We also point out some simple settings under which optimal moving point structures are easy to compute.

1 Introduction

Suppose we want to interconnect a collection of mobile agents, modeled as points in d-dimensional space, in one or more groups to enable a certain collaborative task. For instance, a *minimum spanning tree* is helpful if we want to form a single connected group, a *matching* where we wish to pair each point with another, or a *traveling salesman path* if we want a tour of the points. These are natural problems arising in applications such as multi-robot systems, mobile sensor networks, or a group of human and robotic agents performing collaborative tasks. It is typically desirable that these agents are able to communicate with others, which can be abstracted as a problem of maintaining certain graph structures among a set of moving points, with edge lengths in the graph serving as a natural optimization criterion. The minimum spanning tree, matching, and TSP are classical graph optimization problems for which polynomial time (exact or approximation) algorithms have been known for several decades [7, 6].

When the underlying points are in motion, however, the optimal (or near-optimal) graph structure must be frequently recomputed, which is both inconvenient and resource expensive. There is extensive research in robotics, sensor networks, and computational geometry on how to efficiently detect when the underlying graph structures must be updated and how to update them [4, 9, 5, 1, 8, 2]. Our work is motivated by the recent work of Akitaya et al [3], which explored an alternative approach to maintaining the spanning tree over data in motion. Specifically, if we wanted to choose a *single* spanning tree T for the *entire* motion, which spanning tree would be the best? In other words, which fixed spanning tree topology minimizes the *maximum total length* obtained during the course of the motion? Akitaya et al [3] show that this problem is NP-Hard in the plane and describe a 2-approximation for it.

Our Results

In this paper, we continue the line of research in [3] and explore moving point versions of three classical problems: minimum spanning tree, matching and traveling salesman tour. We show that all of them are NP-hard *even in one dimension*, and *even under highly constrained linear motions.* Second, we establish bounds on the *regret ratio* of these moving point structures as a way to quantify the tradeoff between convenience of a fixed topology and the maximum cost of the structure. Finally, we also point out two natural instances of the moving point structures for which an optimal is easy to find. In the interest of simpler presentation, we focus on the moving point MST in describing our main results, and summarize their extensions to matching and TSP at the end. Our key results can be summarized as follows.

*UC Santa Barbara, {nmwachholz, suri}@cs.ucsb.edu

- The MMST problem is NP-hard even for *unit speed* linear motion in one dimension.
- The problem remains NP-hard in 1D even if all points linearly move away from the origin (radial expansion).
- MMST regret ratio is $\Omega(\sqrt{n})$ even for linear motion in one dimension, but has a tight bound of 2 for *unit speed* motion.
- The MMST problem can be solved in polynomial time, in any fixed dimension d, if (1) all points are moving away from the origin at uniform velocity (uniform expansion), or (2) we want to minimize the *average* cost of the MST during the motion.
- All the hardness results also hold for MMM and MMTSP.

2 Preliminaries

Throughout this paper, we consider points under linear motion, each moving along a straight line with constant speed; different points can move with different speeds unless otherwise specified. Thus, a *moving point* p is a continuous function $p : [0, 1] \to \mathbb{R}^d$, and the distance between two moving points p and q at time t is $\|p(t) - q(t)\|$.

Given a set $S = \{p_0, \ldots, p_{n-1}\}$ of n moving points, we call a spanning tree T of S a *moving spanning tree* whose weight (or length) at time t is $w_T(t) = \sum_{pq\in T} \|p(t) - q(t)\|$. We use $w(T) = \sup_t w_T(t)$ to denote the maximum weight of T during the motion. A *minimum moving spanning tree* (MMST) of S is one with minimum weight, namely,

$$\operatorname*{arg\,min}_{T\in\mathcal{T}(S)} \; w(T),$$

where $\mathcal{T}(S)$ is the set of all possible moving spanning trees of S. Similar definitions hold for a *minimum moving matching* (MMM), where input is a set of $2n$ points and the goal is to find a matching whose maximum weight during the motion is minimized, or *minimum moving traveling salesman tour* (MMTSP), where the goal is to find a spanning path of the points with minimum maximum weight during the motion.

A useful fact about linear motion, as observed in [3], is that the Euclidean distance function $d(t) = \|p(t)-q(t)\|$ is convex, and so the maximum distance between any two points p and q, denoted $|pq|_{\max}$, occurs at an extreme point of the interval $[0, 1]$. That is, $|pq|_{\max} = \max\{\|p(0) - q(0)\|, \|p(1) - q(1)\|\}$. This in turn implies that the weight function w_T is also convex, and so $w(T) = \max\{w_T(0), w_T(1)\}$. We now argue that the MMST is *invariant* under *scaling, translation, and velocity addition*, a fact that will be crucial to some of our proofs.

Let $S = \{p_0, \ldots, p_{n-1}\}$ be a set of moving points in d dimensions. Pick any constant scaling factor $\alpha \in \mathbb{R}^+$, velocity vector $\beta \in \mathbb{R}^d$ and offset $\gamma \in \mathbb{R}^d$. For each p_i, define the transformed moving point $p'_i : [0, 1] \to \mathbb{R}^d$ as $p'_i(t) = \alpha p_i(t) + \beta t + \gamma$. Denote the transformed set as $S' = \{p'_i \mid p_i \in S\}$. Every spanning tree of S maps to a corresponding spanning tree of S' in the obvious way. We then have the following claim.

Lemma 1 *MMST is topologically invariant under scaling, added velocity, and translation.*

Proof. Let S be a set of moving points, and let S' be the transformed set under scaling factor α, velocity vector β and translation γ. We show that if T is an MMST of S, then it is also an MMST of S'.

The distance between two points $p, q \in S$ at time t is $\|p(t) - q(t)\|$, while the distance between their transformed images p' and q' is $\|(\alpha p(t) + \beta t + \gamma) - (\alpha q(t) + \beta t + \gamma)\| = \alpha\|p(t) - q(t)\|$. Since the weight of each edge is simply scaled by α, for any spanning tree T of S and its corresponding tree T' for S', we have $w_{T'}(t) = \alpha w_T(t)$, which implies $w(T') = \alpha w(T)$, thus proving the claim. □

The transform is invertible and so the previous lemma also implies the following.

Corollary 2 *An MMST of a transformed set S' is also an MMST of the original set S.*

3 Hardness of MMST on the Line

In this section, we show that computing the MMST of n linearly moving points on the line is NP-hard *even if all points have the same speed.* We then show that a number of other variations are also hard.

We adopt the convention that positive x-axis is the rightward direction, and so points with positive (resp. negative) velocity are moving to the right (resp. left). In particular, since all points have the same speed, the velocity of each point is either $+1$ or -1.

Our reduction uses the well-known NP-hard problem PARTITION, where given n positive integers $a_0, \ldots, a_{n-1}$, we must decide if there is a subset $I \subseteq \{0, \ldots, n-1\}$ such that

$$\sum_{i\in I} a_i \;=\; \frac{1}{2}\sum_{i=0}^{n-1} a_i.$$

Given an instance of PARTITION, we construct an instance of MMST on the line with unit-speed moving points. We simplify the presentation by assuming the velocity of each point is either 0 or -2, which is then easily transformed into unit speeds by adding $+1$ to each velocity without changing its MMST by virtue of Corollary 2. We construct a decision version of the unit speed MMST problem as follows.

Construction 1 *Let $\ell = \max a_i$. For each $i \in \{0, \ldots, n-1\}$, we add the following five moving points to our set S. See Figure 1 for illustration.*

- *A_i stationary at $10i$.*
- *B_i stationary at $10i + 2 - \frac{a_i}{4\ell}$.*
- *C_i stationary at $10i + 2 + \frac{a_i}{4\ell}$.*
- *D_i stationary at $10i + 4$.*
- *E_i moving from $10i + 3$ to $10i + 1$.*

For this input S, we ask if there is a moving spanning tree T with $w(T) \leq 11n - 6$.

Theorem 3 *The decision version of the MMST problem is NP-Hard for unit speed points on a line.*

Proof. Let $S_i = \{A_i, B_i, C_i, D_i, E_i\}$, for each $i \in \{0, \ldots, n-1\}$. Let K_0 be the set of edges D_iA_{i+1} for $i \leq n-2$, and define K_1 as K_0 plus the edges joining pairs of points within each S_i, for each $i \leq n-1$. Observe that all edges in K_1 have length ≤ 6 at all times, and any edge not in K_1 has length > 6 at all times.

We claim that there exists a moving minimum spanning tree T whose edges are all in K_1. Assume the contrary, and let T' be an MMST containing an edge $e \notin K_1$. Removing this edge from T' disconnects the tree into two components, which can be rejoined by an edge in K_1, contradicting the minimality of T'. Therefore we can assume that there is an MMST T with all edges in K_1.

Each edge in K_0 is a bridge of the graph (S, K_1) and therefore must be included in T. Each of these bridges connects two components, where each component is some moving spanning tree T_i on S_i.

We argue that each T_i has the following form: it contains the path $A_iB_iC_iD_i$, with E_i connected either to B_i (in which case $w_{T_i}(0) = 5 + a_i/4\ell$ and $w_{T_i}(1) = 5 - a_i/4\ell$) or connected to C_i (which flips these weights). The other trees are all easily seen to be sub-optimal; in particular, both $A_iE_iB_iC_iD_i$ and $A_iB_iC_iE_iD_i$ have cost at least $6 + a_i/2\ell$ at either the start or the end, while $A_iB_iE_iC_iD_i$ has cost at least $6 - a_i/2\ell$ at both start and end. All other trees are trivially suboptimal.

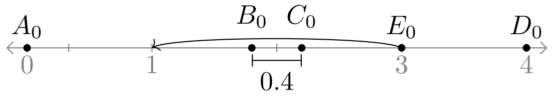


Figure 1: The points in S_0 when $a_0 = 8$ and $\ell = 10$. The optimal trees include the path $A_0B_0C_0D_0$, with E_0 connecting to either B_0 or C_0.

Define $I = \{i : E_i \text{ connects to } B_i\}$, and let $I' = \{0, \ldots, n-1\} - I$. This allows us to write the weights of T as

$$\begin{aligned} w_T(0) &= w(K_0) + \sum_{i \in I} w_{T_i}(0) + \sum_{i \in I'} w_{T_i}(0) \\ &= 6(n-1) + \sum_{i \in I} \left(5 + \frac{a_i}{4\ell}\right) + \sum_{i \in I'} \left(5 - \frac{a_i}{4\ell}\right) \\ &= 11n - 6 + \sum_{i \in I} \frac{a_i}{4\ell} - \sum_{i \in I'} \frac{a_i}{4\ell} \end{aligned}$$

and by symmetry

$$w_T(1) = 11n - 6 - \sum_{i \in I} \frac{a_i}{4\ell} + \sum_{i \in I'} \frac{a_i}{4\ell}.$$

The maximum weight of T is

$$w(T) = 11n - 6 + \left| \sum_{i \in I} \frac{a_i}{4\ell} - \sum_{i \in I'} \frac{a_i}{4\ell} \right|,$$

which achieves its minimum value of $11n - 6$ when

$$\sum_{i \in I} \frac{a_i}{4\ell} = \sum_{i \in I'} \frac{a_i}{4\ell} \iff \sum_{i \in I} a_i = \sum_{i \in I'} a_i = \frac{1}{2} \sum_{i=0}^{n-1} a_i.$$

Thus, if $w(T) \leq 11n - 6$ holds, then the set I is also a solution to PARTITION. This completes the proof. $\square$

Naturally, the MMST problem is also NP-hard for points moving on the line with *arbitrary* but constant speeds. This can also be shown by a simple modification of the construction in [3] used to show the hardness for points moving in two dimensions. We omit that simple construction, and simply state the result below, which is then used to show additional hardness results.

Theorem 4 *The decision version of the MMST problem is NP-Hard for points moving on a line with constant but arbitrary speeds.*

We next show that the problem remains NP-hard under the bounded speed assumption, where the ratio between the maximum and the minimum speeds is upper bounded by some constant $c > 1$. In particular, let $v_i = p_i(1) - p_i(0)$ be the velocity of p_i, where as before positive velocity means rightward motion. Let $s_i = \|v_i\|$ be the speed of p_i, where we assume that $s_i > 0$ for all i, and that the ratio of max to min speeds is bounded:

$$\frac{\max s_i}{\min s_i} \leq c$$

Lemma 5 *MMST problem on a line under bounded speed linear motion is NP-hard.*

Proof. We reduce the MMST problem on the line with arbitrary speeds to our bounded speed ratio problem. First, let $v_{\min} = \min v_i$ and let $v_{\max} = \max v_i$. We transform the input set of moving points S into S' by first adding the velocity $-v_{\min}$ to each point. This shift makes the new minimum velocity 0 and the new maximum velocity $v_{\max} - v_{\min}$.

Next, we add $(v_{\max} - v_{\min})/(c-1) > 0$ to each point's velocity, which ensures that the ratio between the maximum and the minimum speeds is

$$\frac{\max s'_i}{\min s'_i} = \frac{v_{\max} - v_{\min} + (v_{\max} - v_{\min})/(c-1)}{(v_{\max} - v_{\min})/(c-1)} = c.$$

Thus the transformed input S' has bounded speed ratio, and yet by Corollary 2, the two instances S and S' have the same MMST. This completes the proof. □

Finally, we consider another natural velocity-constraint motion: *radial expansion*, where all points are linearly (with different speeds) moving away from the origin. We show that even under this restricted *big bang* expansion model of motion, the MMST problem remains hard even on the line.

Lemma 6 *MMST problem on a line under radial expansion linear motion is NP-hard.*

Proof. We again reduce the MMST problem on the line with arbitrary speeds to our problem. Given a set of moving points $S = \{p_0, \ldots, p_{n-1}\}$ on the line, let $v_i = p_i(1) - p_i(0)$ be the velocity of p_i. Let $v = \min\{v_i, 0\}$ equal the largest *negative velocity* (leftward speed) or zero. Let $o = \min\{p_i(0)\}$ denote the left-most position in S at the start of the motion. We transform our input instance S into S' by adding velocity $-v$ to all points and translating their positions by $-o$.

We now claim that the moving points in S' satisfy radial expansion. This follows because all points in S' have been shifted to the right of the origin, and none of the points have negative velocity, meaning they are all moving to the right. Because the transformation is addition of velocity and translation, by Corollary 2, MMST remains invariant, which completes the proof. □

4 Regret Ratio of Moving Spanning Trees

The MMST problem is motivated by the attractiveness of keeping a fixed topology throughout the motion, and among all spanning trees, MMST is the one with the smallest maximum weight during the motion. So, how much worse is the MMST compared to a *kinetic structure maintaining a minimum spanning tree* throughout the motion? We quantify this tradeoff using the worst-case ratio between the two.

Given a set S of moving points, let $\mathcal{T}(S)$ be the set of all spanning trees on S. Let K be a kinetic minimum spanning tree, meaning $w_K(t) = \min_{T^* \in \mathcal{T}(S)} w_{T^*}(t)$, the weight of a minimum spanning tree at time t. Then $w(K) = \sup_t w_K(t)$ is the maximum cost of a kinetic minimum spanning tree during the motion of S.

For any fixed spanning tree T of S, we define its *regret ratio* as $r(T) = w(T)/w(K)$. Clearly, among all fixed trees, an MMST has the minimum regret. We now show bounds on MMST's regret ratio for moving points in one dimension.

4.1 Regret Ratio for Arbitrary Speed Linear Motion

Theorem 7 *MMST regret is at least $\Omega(\sqrt{n})$ for n moving points on the line.*

Proof. For any $b \geq 1$, we can construct a set S of $O(b^2)$ moving points on the line where $r(T) \geq b$ for any spanning tree T of S. Setting $b = \sqrt{n}$ establishes the claim. Our construction works as follows.

Let p_{ij} denote a moving point such that $p_{ij}(0) = i$ and $p_{ij}(1) = j$. That is, p_{ij} moves from i to j. Let $k = 2\lceil b \rceil$, and choose S as the set of all p_{ij}, where $0 \leq i, j \leq k$.

We now show that any tree on S has regret at least $k/2 = \lceil b \rceil \geq b$. To aid analysis, we interpret the set S of 1D moving points as a set of 2D stationary points, where a point p is placed at $P = (p(0), p(1))$. Thus, in our construction, each p_{ij} maps to $P_{ij} = (i, j)$. See Figure 2 for illustration.

Now, consider any tree T on these points. At the start, the weight of T is the sum of the *horizontal* components of the edges of T. At the end, the weight is the sum of the *vertical* components. Suppose there are h edges with a non-zero horizontal component, and v edges with a non-zero vertical component. The weight of each of these non-zero components is at least 1, and so $w_T(0) \geq h$ and $w_T(1) \geq v$. Furthermore, $h + v \geq (k+1)^2 - 1$, the number of edges in T. Finally, because the moving points always lie between p_{00} and p_{kk} on the line, we have $w(K) = k$. This gives that

$$r(T) = \frac{w(T)}{w(K)} \geq \frac{\max\{v, h\}}{k} \geq \frac{(k+1)^2 - 1}{2k} \geq \frac{k}{2} \geq b.$$

□

4.2 Regret Ratio for Unit Speed

Complementing the previous lower bound on the regret ratio, we show that for unit speed motion, the regret ratio has a tight bound of 2.

Theorem 8 *Regret ratio of an MMST for unit speed moving points on a line is at most 2, and this bound is tight.*

Proof. Partition S into the set of leftwards-moving points S_l and rightwards-moving points S_r. Let P_l be a

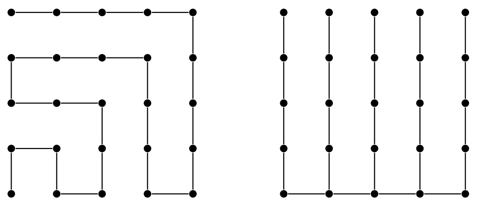


Figure 2: The 2D interpretation of the construction for $b = 2$, along with two sample spanning trees. The tree on the right results by connecting the points in the left-to-right order of their initial positions.

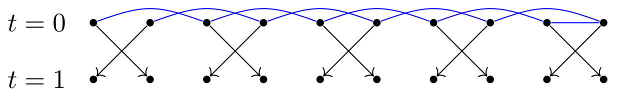


Figure 3: A set of unit speed moving points. Any MMST on this construction will have regret approaching 2 as the number of points increases. One possible MMST is shown in blue.

path connecting S_l in order and let P_r be a path connecting S_r in order. Let e be an edge between the right-most point of P_l and the right-most point of P_r. Then $T^* = P_l \cup \{e\} \cup P_r$ is a moving spanning tree on S. Let $\ell = w(K)$ be the maximum distance the points span throughout the motion.

It's easy to see that at all times either $w_{P_l \cup \{e\}}(t) \leq \ell$ and $w_{P_r}(t) \leq \ell$, or $w_{P_r \cup \{e\}}(t) \leq \ell$ and $w_{P_l}(t) \leq \ell$. Thus $w_{T^*}(t) = w_{P_l}(t) + w_{\{e\}}(t) + w_{P_r}(t) \leq 2\ell$, meaning $r(T^*) \leq 2$. The weight of an MMST is no greater, so its regret is also bounded by 2. Figure 3 shows a construction proving that this bound is tight. □

5 Some Tractable Cases of Moving Spanning Trees

In this section, we complement the negative (hardness) results of the previous section with some natural models of motion or cost for which optimal is easy to compute.

5.1 Motion with Uniform Expansion

Our first result complements Lemma 6: if the motion is radial expansion (big bang model) but all points move away from the origin at the *same constant speed* then MMST is easy to compute. In fact, this holds for any fixed dimension $d \geq 1$.

We say that a set $S \in \mathbb{R}^d$ of moving points is under *uniform expansion* if all points in S move away from the origin at a constant speed $c > 0$. That is, if the start position of a point $p \in S$ is at $p(0) \neq 0$, then its end position is at $p(1) = p(0) + c\frac{p(0)}{\|p(0)\|}$. We have the following easy result.

Lemma 9 *Let S be a set of n moving points under uniform expansion in d dimensions. Then, a minimum spanning tree of S at $t = 1$ (end of the motion) is also an MMST of S.*

Proof. Consider any pair of points $p, q \in S$, and let θ, where $0 \leq \theta \leq \pi$, be the angle between $p(0)$ and $q(0)$ with the respect to the origin. During the motion, the distance between the points will increase by $\sqrt{c^2 + c^2 - 2c^2 \cos\theta} = c\sqrt{2}\sin(\theta/2) \geq 0$ (see Figure 4). Since the distance monotonically increases, we must have $|pq|_{\max} = \|p(1) - q(1)\|$.

It follows that, for any moving spanning tree T, the weight of every edge in T achieves its maximum at $t = 1$, and so $w(T)$ also achieves its maximum at that time. Therefore, a minimum weight spanning tree at end of the motion $t = 1$ is also an MMST of the set S. □

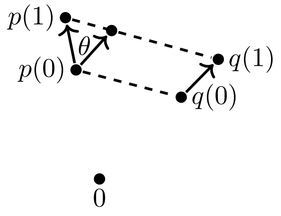


Figure 4: The distance between two points never decreases if they are under uniform expansion.

Since the minimum spanning tree of n points in any fixed dimension d can be computed in polynomial time, the MMST problem for uniform expansion is tractable.

In fact, Lemma 9 holds for any restriction that causes the distance between any two points to be non-decreasing over the motion. For example, *proportional expansion* requires that each point p move from $p(0)$ to $p(1) = ap(0)$ for some constant $a \geq 1$. An MMST of points under proportional expansion can be found by calculating an MST at the end of their motion, using the same reasoning as above.

5.2 Minimum Average Weight

The MMST problem minimizes the *maximum* weight of the spanning tree at any time during the motion. A different but also sensible objective might be the *total weight* of a moving spanning tree, integrated over the entire motion. For instance, if tree length is a measure of resource consumption, then this reflects the total consumption during the motion. Since we have normalized the motion duration to unit interval $[0, 1]$, this is also equivalent to the *average* weight of the moving spanning tree:

$$\overline{w}(T) = \int_0^1 w_T(t)\, dt.$$

Unlike MMST, computing a moving spanning tree with minimum average weight (MAMST) is easy.

Lemma 10 *For a set S of n moving points in d dimensions, an MAMST of S can be found in $O(n^2)$ time, for any fixed d.*

Proof. Let S be a set of n moving points. For any pair $p, q \in S$, define $|pq|_{\text{avg}} = \int_0^1 \|p(t) - q(t)\| \, dt$. We can then rewrite the average weight of a tree T on S as $\overline{w}(T) = \sum_{pq \in T} |pq|_{\text{avg}}$. Now construct a complete graph G with S as the vertex set, and weight of each edge pq set to $|pq|_{\text{avg}}$. The graph G can be constructed in $O(n^2)$ time because each $|pq|_{\text{avg}}$ can be calculated in constant time.

It is now easy to see that the an MST of G is an MAMST of the moving points, and an MST of G can be found in $O(n^2)$ time using Prim's algorithm. □

In [3], it was shown that the minimum bottleneck moving point spanning tree (MBMST) can be computed efficiently, and our average weight spanning tree adds another natural easy-to-compute variant for moving points. (Recently, Wang et al. [10] presented a subquadratic algorithm for the MBMST, improving the $O(n^2)$ bound of [3]. It is an interesting open question whether a similar time bound is also possible for MAMST.)

6 Moving Point Matching and TSP

In this section, we briefly sketch the constructions for proving NP-hardness of matching and TSP, and state without proof their regret bounds. Technical details are similar to those in MMST, and omitted from this extended abstract.

6.1 Minimum Moving Matching

Let S be a set of $2n$ moving points. A *moving matching* M is a set of n edges such that each point is matched with exactly one other. The weight of M is the *maximum* weight of M throughout the motion.

We show that the problem of finding a minimum moving matching (MMM) is NP-Hard *even for unit speed moving points on the line.* The construction is similar to the one from Theorem 3, and illustrated in Figure 5.

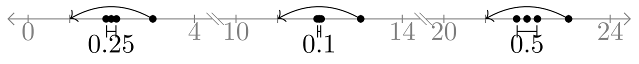


Figure 5: The matching instance corresponding to the PARTITION input $(a_0, a_1, a_2) = (5, 2, 10)$. The PARTITION problem has a solution if and only if there is an MMM with weight at most $n + \frac{1}{4\ell} \sum_{i=0}^{n-1} a_i$.

The regret of an MMM is unbounded in general, using a similar construction to Theorem 7. We do not have a non-trivial upper bound for the MMM regret under unit speed motion, but can show that it is *strictly larger than* 2. Specifically, we show a construction (see the Appendix) with regret ratio of 11/5.

6.2 Minimum Moving TSP

Let S be a set of n moving points. A *moving traveling salesman path* P is a path of $n - 1$ edges that visits every point in S. The weight of P is the *maximum* weight of the path throughout the motion. The construction shown in Figure 6 is used to prove that the problem of finding the minimum moving traveling salesman path (MMTSP) is NP-Hard even for unit speed moving points on the line.

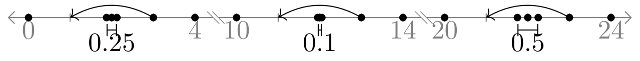


Figure 6: The TSP instance corresponding to the PARTITION input $(a_0, a_1, a_2) = (5, 2, 10)$. The PARTITION problem has a solution if and only if there is an MMTSP with weight at most $12n - 6$.

We also can show that regret ratio of an MMTSP is unbounded in general, but is bounded by 2 for unit speed motion, using constructions similar to Theorem 7 and Theorem 8.

7 Concluding Remarks

We explored several classical geometric problems under the moving point model of [3], and showed that they remain NP-hard even in one dimension and even under highly constrained motions. We did not discuss approximation algorithms but a 2-approximation of MMST is easily computed in $O(n \log n)$ time using the approach of [3], namely, map the 1D moving points into 2D stationary points and compute their MST under the L_1 norm.

We also analyzed the regret ratio of these structures, showing that even in one dimension this can be unbounded in general, but is modest for unit speed. Finally, we suggested two simple settings (uniform expansion and minimum average weight) where the optimal structures are easy to compute.

A number of open problems are suggested by our work. First, it will be interesting to derive a non-trivial upper bound on the regret ratio of these structures in higher dimension under *unit speed motion.* (Proving a tight bound for the regret of moving point matching in one dimension is also an interesting problem.) Second, without the unit speed restriction, it will also be interesting to bound the regret ratio if we are allowed to update the structure k times. In particular, how large must k be to guarantee a constant regret ratio?

References

[1] P. K. Agarwal, D. Eppstein, L. J. Guibas, and M. R. Henzinger. Parametric and kinetic minimum spanning trees. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 596–605. IEEE Computer Society, 1998.

[2] P. K. Agarwal, L. J. Guibas, H. Edelsbrunner, J. Erickson, M. Isard, S. Har-Peled, J. Hershberger, C. S. Jensen, L. E. Kavraki, P. Koehl, M. C. Lin, D. Manocha, D. N. Metaxas, B. Mirtich, D. M. Mount, S. Muthukrishnan, D. K. Pai, E. Sacks, J. Snoeyink, S. Suri, and O. Wolfson. Algorithmic issues in modeling motion. *ACM Comput. Surv.*, 34(4):550–572, 2002.

[3] H. A. Akitaya, A. Biniaz, P. Bose, J. D. Carufel, A. Maheshwari, L. F. S. X. da Silveira, and M. Smid. The minimum moving spanning tree problem. *J. Graph Algorithms Appl.*, 27(1):1–18, 2023.

[4] M. J. Atallah. Dynamic computational geometry (preliminary version). In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 92–99. IEEE Computer Society, 1983.

[5] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT press, 2009.

[7] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry. In *Computational geometry.* Springer, 1997.

[8] C. S. Helvig, G. Robins, and A. Zelikovsky. The moving-target traveling salesman problem. *J. Algorithms*, 49(1):153–174, 2003.

[9] C. L. Monma and S. Suri. Transitions in geometric minimum spanning trees. *Discret. Comput. Geom.*, 8:265–293, 1992.

[10] H. Wang and Y. Zhao. Computing the minimum bottleneck moving spanning tree. In S. Szeider, R. Ganian, and A. Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, pages 82:1–82:15, 2022.

Appendix

We provide an example where the regret ratio of a minimum moving point matching (MMM) is strictly larger than 2 under unit speed motion on a line. For clarity, our points are transformed with added velocity +5 using Corollary 2. Construct the following 6 points, which are illustrated in Figure 7:

- p_0, p_1, p_2 moving from $0, 1, 2$ to $10, 11, 12$; and
- q_0, q_1, q_2 static at $3, 6, 9$.

Consider a kinetic minimal matching. In the initial positions, the matching (p_0p_1, p_2q_0, q_1q_2) is optimal, having weight 5. With a few edge changes during the motion, it can maintain this cost (or less), eventually ending as the matching (q_0q_1, q_2p_0, p_1p_2). Therefore $w(K) = 5$.

On the other hand, consider the MMM (p_0q_0, p_1p_2, q_1q_2). It has initial cost 11, and final cost 7. There is a similar MMM (p_0p_1, p_2q_2, q_0q_1) with these costs reversed. But this is as good as we can do. So the MMM regret is $11/5 = 2.2 > 2$.

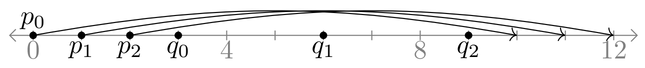


Figure 7: A set of 6 unit speed moving points (transformed for clarity to have velocities 0 and +10). An MMM on these points has regret $11/5 = 2.2$.

Online Square Packing with Predictions[*]

Stephane Durocher[†] Shahin Kamali[‡] Pouria Zamani Nezhad[§]

Abstract

Square packing is a geometric variant of the classic bin packing problem, which asks for the placement of squares of various lengths into a minimum number of unit squares. In this work, we study the online variant of the problem in which the input squares appear sequentially, and each square must be packed before the next square is revealed. We study the problem under the *prediction setting*, where the online algorithm is enhanced with a potentially erroneous prediction about the input sequence. We design an online algorithm that receives predictions concerning the sizes of input squares and analyze its *consistency* (the competitive ratio assuming no error in the predictions) and *robustness* (the competitive ratio under adversarial error). In particular, our algorithm has consistency $1.77\bar{9}$ and robustness at most 5.89. These results show improvements over the best previous algorithm [24], designed for perfect predictions, with a consistency of 1.84 and a robustness of at least 21.

1 Introduction

Given a multiset of n square items, each with a fixed sidelength in $(0, 1]$, the square packing problem seeks to assign each item to a unit square bin, such that the number of bins is minimized. We consider orthogonal packings, in which each item's interior is contained in the interior of its assigned bin, each item's edges are oriented parallel to its assigned bin's edges (axis parallel), and no two items' interiors in the same bin intersect (pairwise interior-disjoint). We refer to a square's sidelength as its *size*. This is a geometric variant of the classic bin packing problem. Similarly to the bin packing problem, square packing is NP-hard, but admits an Asymptotically Polynomial-Time Approximation Scheme (APTAS) [10].

We consider *online square packing*, in which input square items are revealed one at a time in an online sequence. Upon receiving each item, an algorithm must assign it to a bin with sufficient space immediately, without any knowledge about future items. Bin assignments are irrevocable. The standard measure for evaluating an online algorithm is the *asymptotic competitive ratio*, which compares the cost of the online algorithm against the optimal (offline) cost in the worst case. For the online square packing problem, the asymptotic competitive ratio of an online algorithm $\textsc{Alg}$ is

$$\lim_{n\to\infty} \sup_{\sigma:|\sigma|=n} \frac{|\textsc{Alg}(\sigma)|}{|\textsc{Opt}(\sigma)|},$$

where $|\textsc{Alg}(\sigma)|$ denotes the number of bins used by $\textsc{Alg}$ to pack the input sequence σ, and $|\textsc{Opt}(\sigma)|$ denotes the minimum number of bins required by any (optimal) packing of σ. We refer to asymptotic competitive ratio simply as *competitive ratio*. No online square packing algorithm can achieve a competitive ratio better than 1.75 [8], while the best previous algorithm has a competitive ratio of at most 2.0885 [20].

Square packing has been studied under the *advice setting*, which relaxes the assumption that the algorithm has no advance information about the input sequence, and provides the online algorithm access to error-free information about the input sequence called *advice* before packing the first item in the input sequence [18]. The objective is to quantify trade-offs between the competitive ratio and the number of bits of advice. For square packing, there is an online algorithm that achieves a competitive ratio of at most 1.84 with $O(\log n)$ bits of advice [24]. Unfortunately, this result has little practical significance, partially because the advice is assumed to be error-free.

In this paper, we study the online square packing problem under a recently developed and more practical model, which seeks to leverage *predictions* about the input sequence [28]. Specifically, the algorithm can access some machine-learned information about the input sequence. Unlike with advice, predictions may be erroneous. Moreover, the predictions should be *efficiently learnable* (e.g., via sampling the input sequence). The objective is to design an algorithm that performs well if the prediction is accurate while maintaining a good competitive ratio even when the prediction is highly erroneous (i.e., adversarial). We refer to the competitive ratio of an online algorithm with an error-free prediction as its *consistency* and to the competitive ratio with an adversarial prediction as its *robustness* [28]. Several online optimization problems have been stud-

[*]This work is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[†]University of Manitoba, stephane.durocher@umanitoba.ca
[‡]York University, kamalis@yorku.ca
[§]University of Manitoba, zamaninp@myumanitoba.ca

ied under the prediction model, including bin packing [3, 4], scheduling [2, 12, 30, 32], knapsack [11, 23, 33], caching [28, 31], matching [5, 25, 26], and various graph problems [7, 9, 14, 15, 19]. See also the survey by Mitzenmacher and Vassilvitskii [29] and the collection at [1].

1.1 Contribution

We study the square packing problem under a setting where the online algorithm exploits natural predictions concerning the *frequency* of item sizes. We classify square items based on their sizes and consider predictions on the number of items within certain classes. To be more precise, predictions specify the number of items in the input sequence with side-lengths in the ranges $(2/3, 4/5]$, $(3/5, 2/3]$, $(11/20, 3/5]$, $(1/2, 11/20]$, and $(1/3, 2/5]$. For an input sequence of n items, these predictions can be encoded in $O(\log n)$ bits, and they are Probably Approximately Correct (PAC)-learnable [13]. We design an algorithm, named Reserve-and-Pack (RAP), which makes use of the above predictions. Our results can be summarized as follows:

- We show that RAP has a consistency of $1.77\bar{9}$ (Theorem 6). In other words, when predictions are error-free (they are advice), the competitive ratio of RAP is at most $1.77\bar{9}$. This result is an improvement over the algorithm of [24], Almost-Online-Square-Packing (AOSP), which has a consistency of 1.84. Both algorithms use a prediction (advice) of size $O(\log n)$.
- We show that the robustness of AOSP is at least 21 (Theorem 7). Moreover, we prove that the robustness of RAP is at most $100/17 \lessapprox 5.89$ (Theorem 8). In other words, RAP dominates AOSP regarding both consistency and robustness. This is due to its improved item classification and increased flexibility in adapting to patterns in the input rather than overly relying on the predicted patterns.

2 Reserve and Pack (RAP) Algorithm

In this section, we present our algorithm Reserve-And-Pack (RAP). RAP works by classifying items based on their sizes and receiving predictions about the frequency (number) of items from certain classes with larger sizes. The algorithm proceeds by reserving a *placeholder* for each of these items in anticipation of their arrival. We point out that AOSP receives similar predictions and also uses placeholders [24]. RAP improves over AOSP by refining the item classification, which results in improved consistency. In addition, RAP only reserves space for certain items of larger size, unlike AOSP, which forms an offline packing of the predicted input and reserves placeholders for *all* items (except for "tiny" items). In other words, RAP's reliance on prediction is minimal compared to AOSP. As a result, it has superior robustness in the case of erroneous predictions.

Class	Interval	Class	Interval
$1a$	$(4/5, \quad 1]$	$2a$	$(2/5, \quad 1/2]$
$1b$	$(2/3, \; 4/5]$	$2b$	$(1/3, \; 2/5]$
$1c$	$(3/5, \; 2/3]$	3–29	$(1/i+1, 1/i]$
$1d$	$(11/20, 3/5]$	30	$(0, \quad 1/30]$
$1e$	$(1/2, 11/20]$		

Table 1: Item classification used by RAP

Item Classification. RAP classifies items into 30 classes based on their sizes. For $i \in [1..29]$, items with size in the range $(\frac{1}{i+1}, \frac{1}{i}]$ belong to class i. Items with sizes in the $(0, 1/30]$ form the 30th class and are called *tiny* items. Items of Class 1, which are larger than 0.5, are called *large* items and are further divided into 5 subclasses $1a, 1b, 1c, 1d$, and $1e$ with sizes corresponding to the intervals $\left(\frac{4}{5}, 1\right]$, $\left(\frac{2}{3}, \frac{4}{5}\right]$, $\left(\frac{3}{5}, \frac{2}{3}\right]$, $\left(\frac{11}{20}, \frac{3}{5}\right]$, and $\left(\frac{1}{2}, \frac{11}{20}\right]$, respectively. Similarly, items of Class 2 are called *medium* items and are further divided into two subclasses $2a$ and $2b$ with respective associated intervals $\left(\frac{2}{5}, \frac{1}{2}\right]$ and $\left(\frac{1}{3}, \frac{2}{5}\right]$. Table 1 summarizes defined classes and their corresponding size intervals.

In addition to items, each bin of RAP has a type, which is determined by the class of items it contains. Specifically, *LM-bins* contain a large or a medium item, say of type ℓ, and smaller items of the same type $t \geq 3$, in which case the bin is referred to as a $\langle \ell, t \rangle$ bin. For example, when $\ell = 2b$ and $t = 10$, the LM-bin is of type $\langle 2b, 10 \rangle$ and only contains medium items of type $2b$ and small items of type 10 (see Figure 1c). When the large item is of type $\ell = 1e$ and the small item is of type $t = 4$, an LM-bin is called a *critical bin* and is allowed to contain items of a third type t', in which case it is referred to as a $\langle 1e, 4, t' \rangle$ bin. In addition to LM-bins, RAP maintains *harmonic* bins that only include items of the same type, say t, in which case the bin is said to be a harmonic-t bin. A harmonic-t bin is said to be a *large harmonic bin* if t is a large or medium type and *small harmonic bin* otherwise. We note that large harmonic bins may change their type to become LM-bins (when a small or tiny item is placed in them).

Preprocessing. RAP relies on predictions about the number of items belonging to large and medium types. Specifically, RAP uses a frequency vector $\mathbf{f} = \langle f_{1b}, f_{1c}, f_{1d}, f_{1e}, f_{2b} \rangle$, where f_t is the predicted number of class t items in the input sequence σ. Note that the predictions do not concern $1a$ and $2a$ items as they are "easy to pack"; i. e., they can be packed into almost full bins without involving other items, as will be clarified later. Before packing the input sequence, for each predicted frequency f_t, RAP creates f_t *placeholders* of class t, that is, a reserved space of equal size to the maximum size of class t items. RAP assigns placeholders of each class in separate bins, while groups of four $2b$ placeholders share one bin. These bins are "virtually" open, and they contribute to the cost of RAP only after an item is placed into them. We assume placeholders are positioned on the top-left of their respective bins.

Online Packing. When possible, RAP places large and medium items in placeholders reserved in the preprocessing step. This is done through a procedure called ASSIGNLM that we will describe shortly. Small and tiny items, however, are packed into designated *containers* that are formed and placed in an online manner. A container is a dedicated space that can accommodate either a single small item or multiple tiny items. Upon the arrival of a small or tiny item of class c, it is placed in an available container of the same class using the corresponding ASSIGNSMALL or ASSIGNTINY procedures, which will also be described later. If no such container exists, RAP creates a new set of class c containers using a subroutine called RESERVE.

The RESERVE subroutine, for any given class $c \geq 3$ (small or tiny), first attempts to place containers of class c in a critical bin, and if not possible, a large harmonic bins using the *L-shape tiling* of [24]. This involves placing containers of type c in the non-reserved space of the bin in a greedy manner in columns and rows that collectively form an "L"-shape. If no critical or large harmonic bin is available, it opens a new small harmonic bin of type c. More precisely, RESERVE takes the following steps to create new containers of type c:

1. If $c \geq 5$ and there is a critical bin B (i.e., a bin of type $\langle 1e, 4\rangle$), add containers of type c to B, using L-shape tiling, and update the type of B to $\langle 1e, 4, c\rangle$.
2. If a large harmonic bin B of type ℓ with a reserved space of r is available, and $1/c \leq 1 - r$, use L-shape tiling to place containers of class c to B, and update the type of B to $\langle \ell, c\rangle$. Here, "available" means that B does not contain any other containers.
3. Otherwise, open a new harmonic bin of type c and place c^2 containers of class c into it.

RAP consists of three main components: ASSIGNLM, ASSIGNSMALL, and ASSIGNTINY, packing corresponding items of large/medium, small, and tiny classes.

- ASSIGNLM assigns each $1a$, or four $2a$ items into a single bin. In addition, it packs an item of class $c \in \{1b, 1c, 1d, 1e, 2b\}$ into any available placeholder of class c. If no placeholder is available (due to a prediction error), it opens a new bin and declares it as a large harmonic bin of type c.
- ASSIGNSMALL packs small items of class c into the next empty container of size $1/c$. If no such container is available, it creates a new set of containers by invoking the RESERVE subroutine.
- ASSIGNTINY places tiny items into *tiny containers*. A tiny container is of size $1/5$ and is dedicated to tiny items. As before, if no tiny container exists, ASSIGNTINY first creates a new set of tiny containers by calling RESERVE. We borrow the algorithm of [22] to pack tiny items into tiny containers. The algorithm repeatedly splits tiny containers into smaller sub-containers to pack a tiny item of size s into a sub-container of size $1/2^k$, where k is the largest integer such that $s \leq 1/2^k$.

Figure 1 shows examples of bins packed by RAP. In particular, Figures 1a and 1e are bins that used to be critical and had their types changed after receiving additional small containers.

3 Consistency Analysis

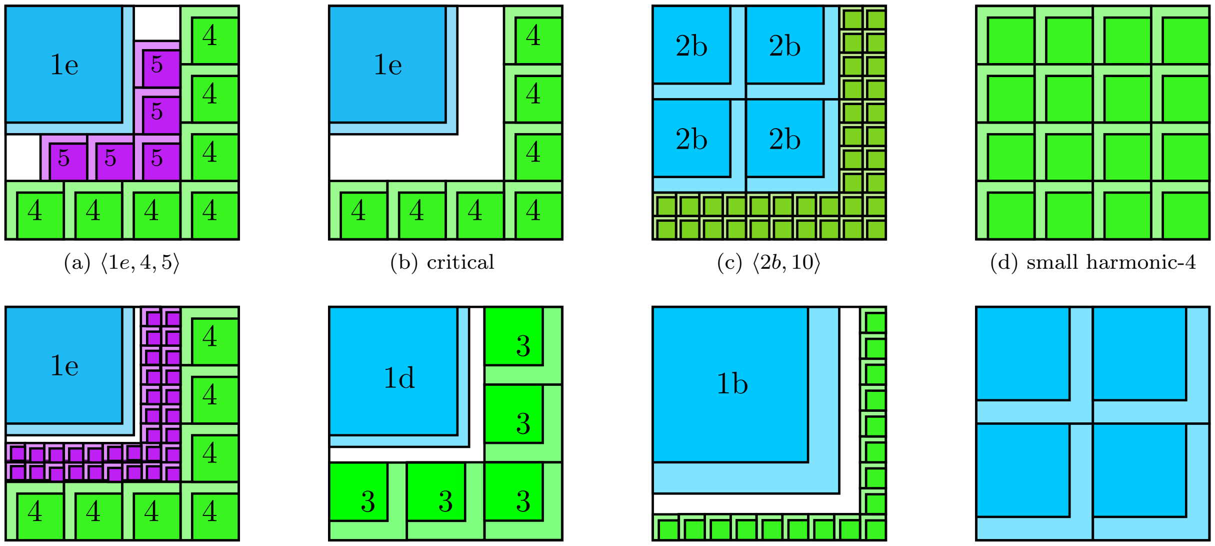
Overview. In this section, we analyze the consistency of RAP. First, we use the following lemma to show that all tiny containers, except possibly the last one, are almost full.

Lemma 1 *[22] Consider the square packing problem where all items are smaller than or equal to $1/M$ for some integer $M \geq 2$. There is an online algorithm that creates a packing in which all bins, except possibly a constant number of them, have an occupied area of size at least $(M^2-1)/(M+1)^2$.*

In our context, tiny items of size at most $1/30$ are packed into containers of size $1/5$. With a scaling argument, the above result applies with $M = 6$. Another ingredient in our proof is a lower bound for the number of containers of a given class placed into a bin using L-shape tiling. In particular, we will use the following lemma:

Lemma 2 *Consider a square space S of sidelength $s \in \{0.75, 1\}$, from which a square space of size $r < s$ is reserved. It is possible to pack $2ki - k^2$ containers of size c in the remainng area of S, where $i = \lfloor s/c \rfloor$ and $k = \lfloor (s-r)/c \rfloor$.*

Proof. Consider an empty bin of size s; it fits i^2 containers of size c, where $i = \lfloor s/c \rfloor$. However, $(i-k)^2$ of these containers overlap with the reserved space, where



(a) $\langle 1e, 4, 5\rangle$ (b) critical (c) $\langle 2b, 10\rangle$ (d) small harmonic-4

(e) $\langle 1e, 4, 12\rangle$ (f) $\langle 1d, 3\rangle$ (g) $\langle 1b, 9\rangle$ (h) large harmonic-$2a$

Figure 1: Examples of possible bin types of Rap. The light-colored areas represent reserved spaces, while dark-colored areas show the minimum occupied area by items of each specified type. Blue areas are placeholders, while green and purple areas are containers. Purple containers are extra containers added to critical bins.

$k = \lfloor (s-r)/c \rfloor$. It implies that there is room to add

$$\begin{aligned}&\lfloor s/c \rfloor^2 - (\lfloor s/c \rfloor - \lfloor (s-r)/c \rfloor)^2\\ &= i^2 - (i-k)^2\\ &= k(2i-k)\\ &= 2ki - k^2\end{aligned}$$

containers to this bin which completes the proof. □

When using L-shape tiling to place containers in large harmonic bins (packing green containers in Figure 1), we have $s = 1$. When using L-shape tiling to place small containers in critical bins (packing purple containers in the figure), we have $s = 0.75$. Table 2 in Appendix presents the minimum occupied area of all LM bins by applying Lemmas 1 and 2.

To prove an upper bound of $1.77\bar{9}$ for the consistency of Rap, we consider three possibilities for the final packings of Rap. The first case is when all bins in the final packing contain a large or medium item. In other words, no small harmonic bin is opened. In this case, we use a standard *weighting technique* [6] to prove the upper bound for the competitive ratio (Lemma 3). The second case is when the final packing has a harmonic bin of class $c \geq 5$. In this case, we show that all bins are sufficiently full to prove the upper bound (Lemma 4). Finally, the third case concerns situations in which there is no harmonic bin of class $c \geq 5$ in the final packing. In this case, we use a more complicated weighting argument that involves solving an integer program to prove the upper bound (Lemma 5).

Case I: No small harmonic bin. Suppose no bin is opened for containers of tiny or small items. We assign a weight of $w(x)$ to an item of size x, and prove that (i) the total weight of items in any bin of Rap, except possibly a constant number of them, is at least 1, while (ii) the weight of items in any bin of Opt is at most α. Therefore, if W denotes the total weight of all items in the input, we can write $\text{Rap}(\sigma) \leq W + c$, for some constant c, while $\text{Opt}(\sigma) \geq \lceil W/\alpha \rceil$, which yields to a competitive ratio of at most α.

Lemma 3 *Suppose there is no small harmonic bin of class $c \geq 3$, in the final packing of* Rap*. Then, the competitive ratio of* Rap *is at most* 1.75*.*

Proof. We assign to all items of class $c \geq 3$ a weight of 0. Large items have a weight of 1, and medium items have a weight of 1/4. All bins in the final packing of Rap, except possibly two of them, either contain a large item or four medium items. Therefore, all bins opened by Rap have a total weight of at least 1. It follows that $|\text{Rap}(\sigma)| \leq W$, where W is the total weight of all items in σ. On the other hand, a bin of $\text{Opt}(\sigma)$ may contain three medium and one large item, e. g., an item of size $0.5 + \epsilon$ and three items of size $0.5 - \epsilon$. Moreover, no more than one large item and four medium items fit into the same bin, giving a maximum total weight of 1.75 for items in any bin of Opt. Therefore, $|\,\text{Opt}(\sigma)| \geq W/1.75$, resulting in a competitive ratio of at most 1.75 for Rap. □

Case II: There is a small harmonic bin of type $t \geq 5$. Suppose there is at least one small harmonic bin of type $t \geq 5$ in the final packing. We show that all opened bins, except possibly a constant number of them, have an occupied area of at least 9/16, which indeed guarantees a competitive ratio of at most 16/9.

Lemma 4 *Assume there is a harmonic-t bin in the final packing of* Rap*, for some $t \geq 5$. Then, the occupied area in all bins, except possibly a constant number of them, is at least* 9/16.

Proof. We begin by observing that large harmonic bins with an item of Class $1a$ or four items of Class $2a$ have an occupied area of at least $16/25 = 0.64$. Next, we show that the final packing of Rap contains no large harmonic or critical bins. For the sake of contradiction, suppose there is a large harmonic or critical bin B of type ℓ, and note that B receives a placeholder in the preprocessing step. Therefore, before opening small harmonic bins, specifically t-harmonic bins, Reserve must use L-shape tiling to place containers of class t in B. This would change the type of B to $\langle \ell, t\rangle$, contradicting its final type being a large harmonic bin of type ℓ. We conclude that the final packing of Rap only contains LM bins and small harmonic bins.

Note that Reserve procedure adds new containers to one bin upon each call, and it is only invoked once there are no empty containers left for an arrived item. It implies there is at most a constant number of empty containers at any time during the execution of the algorithm. Therefore, harmonic bins of type $t' \geq 3$, except possibly a constant number of them (which have empty containers), each contains t'^2 items of class t' and a minimum occupied area of at least $t'^2/(t'+1)^2 \geq 9/16$. Similarly, by Lemma 1, each tiny bin has a minimum occupied area of $35/49 > 9/16$. Finally, Lemmas 1 and 2 show that all LM-bins, except those with empty containers, each has a minimum occupied area of at least 9/16, as reported in Table 2 [Appendix]. Therefore, all bins in the final packing of Rap, except possibly a constant number of them, have a minimum occupied area of at least 9/16. □

Case III: All small harmonic bins are of Class 3 or 4. We consider the case where there is no small harmonic bin of type $t \geq 5$ in the final packing of Rap, while there is a small harmonic of Type 3 or 4. In this case, there are critical bins in the packing, and not enough small items of class $t \geq 5$ were revealed to cover the "wasted" space in these critical bins. Similar to Case I, we use a weighting argument to prove the following lemma.

Lemma 5 *Assume there is no small harmonic bin of type $t \geq 5$ in the final packing, while there is a small harmonic-t' bin, where $t' < 5$. Then, the competitive ratio of* Rap *is at most* $1.77\bar{9}$.

Proof. We assign a fixed weight to all items of the same class except for tiny items, which receive a weight proportional to their sizes. We define $\mathbf{w} = \langle w_1, \ldots, w_{34}\rangle$, where w_i is the weight assigned to items of (sub)class $i \leq 34$. Additionally, we assign a weight of $w(x) = d \times s(x)^2$ to a tiny item x, where $s(x)$ is the size of x and d is a fixed constant called the *density* of tiny items. The specific weights are specified in Table 3 [Appendix]. These weights are defined in a way to guarantee a total weight of at least 1 for all bins in the final packing, except possibly a constant number of them. We use an integer program to prove an upper bound on the weight of bins in Opt. Let $\mathbf{t} = \langle t_1, \ldots, t_{35}\rangle$ denote the maximum size of items in each class, in decreasing order; that is, for $i \leq 7$, t_i denotes the maximum size of large or medium items of subclass i and, for $j \geq 7$, t_j denote the maximum size of items of class $j-5$. Let x_i denote the number of items of class i in a bin, say B, packed by Opt. To maximize the weight of the items in B, we can write the following integer program:

maximize:

$$\alpha = \sum_{i=1}^{34} w_i \cdot x_i + d \cdot \left(1 - \sum_{i=1}^{34} x_i \cdot t_{i+1}^2\right)$$

subject to:

$$\sum_{i=1}^{34} x_i \cdot t_{i+1}^2 \leq 1 \tag{1}$$

$$\sum_{i=1}^{34} \lfloor t_{i+1} \cdot (u+1) \rfloor^2 \cdot x_i \leq u^2, \quad \forall u \in \{1, \ldots, 60\} \tag{2}$$

$$x_i \geq 0 \text{ and } x_i \in \mathbb{Z} \qquad , \quad \forall i \in \{1, \ldots, 34\}$$

The first component of the objective function is the total weight of all non-tiny items, and the second is an upper bound on the weight of all tiny items in B. Constraint 1 ensures the total area of non-tiny items of B does not exceed 1, and Constraint 2 ensures all items fit into B without overlap. This constraint must hold because, for any integer $u \geq 1$, a bin cannot contain more than u^2 squares of size above $\frac{1}{u+1}$. Figure 2 represents the optimal solution to the integer program resulting in a $1.77\bar{9}$ upper bound on α which completes the proof. □

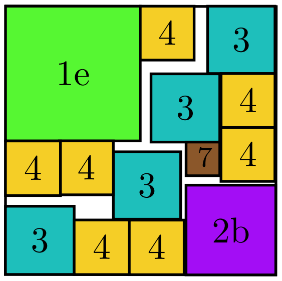


Figure 2: The packing that maximizes the total weight of items in a bin of Opt, as discussed in Lemma 5

Wrapping up. Our results imply the following upper bound for the consistency of Rap.

Theorem 6 Rap *has a consistency of at most* $1.77\bar{9}$.

Proof. Lemmas 3 and 4 show that the competitive ratio is at most $1.77\bar{9}$ in Cases I and III. In case II, Lemma 5 shows that Rap opens at most $16W/9 + c$ bins, where W is the total area of all items, and c is a constant. Given that Opt opens at least W bins, the competitive ratio, in this case, is at most $16/9 < 1.77\bar{9}$. □

4 Robustnes Analysis

In this section, we study the robustness of online square-packing algorithms. We first present a lower bound on the robustness of the previously proposed algorithm, Aosp of [24]. Aosp forms an offline packing of predicted frequency to reserve a placeholder for *all* non-tiny items. Therefore, Aosp is overly reliant on the correctness of the predictions. In particular, one can generate adversarial inputs, formed only by tiny items, in which all placeholders of Aosp remain empty.

Theorem 7 *The* Aosp *algorithm of [24] has a robustness of at least* 21.

Proof. We show that Aosp has a competitive ratio of $841/40 \approx 21$ on an input sequence σ and adversarial frequency predictions $\hat{\mathbf{f}}$, which implies a lower bound of 21 on its robustness.

Aosp forms an offline packing of predicted frequency to reserve a placeholder for *all* expected items, except for "tiny" items. In the context of the Aosp algorithm, tiny items have a size of at most 1/15. As a result, Aosp is overly reliant on the correctness of the predictions. e. g., all placeholders of some class t remain empty if items of that type never arrive.

Given a frequency prediction $\hat{\mathbf{f}}$, encoded by an adversarial oracle, predicting $7n$ small items of size at most $1/4$ and n large items of size at most $3/5$, Aosp forms an offline packing of n bins, where each bin contains one placeholder for a large item, seven placeholders for small items, and forty containers of size at most 1/15 to pack tiny items online.

Consider an input sequence σ_w consist of $40n$ tiny items of size $1/30 + \epsilon$. Given the prediction $\mathbf{f}$, Aosp$(\sigma, \mathbf{f})$ has n bins that are partially filled by tiny items. Opt, however, fits each $29^2 = 841$ of these tiny items into a one bin; therefore, Opt$(\sigma) = 40n/841$. It follows, Aosp$(\sigma, \mathbf{f})$ has a competitive ratio of $841/40 \approx 21$ which completes the proof. □

We now show that Rap has a robustness of at most 5.89. For that, we prove a lower bound for the minimum occupied area of the bins that Rap opens.

Theorem 8 Rap *has a robustness of at most* 5.89.

Proof. The final packing of Rap consists of harmonic, LM, and critical bins. A small harmonic bin of class i has a minimum occupied area of $i^2/(i+1)^2$, which implies a minimum occupied area of 9/16 for all small harmonic bins. Moreover, the occupied area in bins that include large or medium items is at least 1/4.

Next, we consider bins with an empty placeholder. Placeholders for these items were reserved during the preprocessing step and remained empty due to prediction errors. We note that these bins have received non-empty containers with small or tiny items; otherwise, they would be virtually open (do not contain any items), and do not contribute to the final cost. Therefore, the occupied area of LM-bins is at least the minimum total occupied area of the small or tiny items they contain. Table 4 in Appendix shows lower bounds for the occupied area. In particular, the worst-case scenario is realized by the $\langle 1b, 9\rangle$ bins, as shown in Figure 1g, when the placeholder for $1b$-items stays empty and the occupied area is at least 0.17. We can conclude that all bins in the final packing of Rap, except possibly a constant number of them, have a minimum occupied area of at least 0.17, regardless of the quality of predictions. It follows, Rap has a robustness of at most $100/17 \lessapprox 5.89$. □

5 Concluding Remarks

In this paper, we introduced Rap, an online square-packing algorithm that leverages frequency predictions and has superior consistency and robustness over an existing algorithm of [24], Aosp, which overly relies on predictions. The techniques we used for the design of Rap, e.g., differentiating between placeholders and containers, and its analysis, e.g., solving the integer program in Lemma 5, are likely helpful in studying other geometric packing problems under the prediction model such as 2-dimensional box packing [16, 27] and d-dimensional cube packing [17, 21] problems.

References

[1] Algorithms with predictions. `https://algorithms-with-predictions.github.io/`. Accessed: 2023-02-19.

[2] S. Angelopoulos and S. Kamali. Contract scheduling with predictions. *J. Artif. Intell. Res.*, 77:395–426, 2023.

[3] S. Angelopoulos, C. Dürr, S. Jin, S. Kamali, and M. Renault. Online computation with untrusted advice. In *Proc. ITCS*, pages 52:1–52:15, 2020.

[4] S. Angelopoulos, S. Kamali, and K. Shadkami. Online bin packing with predictions. In *Proc. IJCAI*, pages 4574–4580, 2022.

[5] A. Antoniadis, T. Gouleakis, P. Kleer, and P. Kolev. Secretary and online matching problems with machine learned advice. In *Proc. NeurIPS*, 2020.

[6] S. Assmann, D. Johnson, D. Kleitman, and J.-T. Leung. On a dual version of the one-dimensional bin packing problem. *Journal of Algorithms*, 5(4): 502–525, 1984.

[7] Y. Azar, D. Panigrahi, and N. Touitou. Online graph algorithms with predictions. In *Proc. SODA*, pages 35–66, 2022.

[8] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. Lower bounds for several online variants of bin packing. *Theory of Computing Systems*, 63 (8):1757–1780, 2019.

[9] S. Banerjee, V. Cohen-Addad, A., and Z. Li. Graph searching with predictions. In *Proc. ITCS*, volume 251, pages 12:1–12:24, 2023.

[10] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31 – 49, 2006.

[11] J. Boyar, L. M. Favrholdt, and K. S. Larsen. Online unit profit knapsack with untrusted predictions. In *Proc. SWAT*, pages 20:1–20:17, 2022.

[12] J. Boyar, L. M. Favrholdt, S. Kamali, and K. S. Larsen. Online interval scheduling with predictions. In *Proc. WADS*, 2023.

[13] C. L. Canonne. A short note on learning discrete distributions, 2020. arXiv math.ST:2002.11457.

[14] J. Y. Chen, T. Eden, P. Indyk, H. Lin, S. Narayanan, R. Rubinfeld, S. Silwal, T. Wagner, D. P. Woodruff, and M. Zhang. Triangle and four cycle counting with predictions in graph streams. In *Proc. ICLR*, 2022.

[15] J. Y. Chen, S. Silwal, A. Vakilian, and F. Zhang. Faster fundamental graph algorithms via learned predictions. In *Proc. ICML*, volume 162, pages 3583–3602, 2022.

[16] F. R. K. Chung, M. R. Garey, and D. S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3:66–76, 1982.

[17] J. Csirik and G. J. Woeginger. On-line packing and covering problems. In *Online Algorithms, The State of the Art*, volume 1442 of *LNCS*, pages 147–177. Springer, 1996.

[18] S. Dobrev, R. Královič, and D. Pardubská. Measuring the problem-relevant information in input. *RAIRO - Theor. Inf. Appl.*, 43(3):585–613, 2009.

[19] F. Eberle, A. Lindermayr, N. Megow, L. Nölke, and J. Schlöter. Robustification of online graph exploration methods. In *Proc. AAAI*, pages 9732–9740, 2022.

[20] L. Epstein and L. Mualem. Online bin packing of squares and cubes. In *Algorithms and Data Structures*, pages 357–370. Springer International Publishing, 2021.

[21] L. Epstein and L. Mualem. Online bin packing of squares and cubes. In *WADS*, volume 12808, pages 357–370. Springer, 2021.

[22] L. Epstein and R. van Stee. Optimal online algorithms for multidimensional packing problems. *SIAM Journal on Computing*, 35(2):431–448, 2005.

[23] S. Im, R. Kumar, M. M. Qaem, and M. Purohit. Online knapsack with frequency predictions. In *Proc. NuerIPS*, pages 2733–2743, 2021.

[24] S. Kamali and A. López-Ortiz. Almost online square packing. In *Proc. CCCG*, 2014.

[25] T. Lavastida, B. Moseley, R. Ravi, and C. Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In *Proc. ESA*, volume 204, pages 59:1–59:17, 2021.

[26] T. Lavastida, B. Moseley, R. Ravi, and C. Xu. Using predicted weights for ad delivery. In *Proc. ACDA*, pages 21–31, 2021.

[27] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *Eur. J. Oper. Res.*, 141(2):241–252, 2002.

[28] T. Lykouris and S. Vassilvitskii. Competitive caching with machine learned advice. In *Proc. ICML*, pages 3302–3311, 2018.

[29] M. Mitzenmacher and S. Vassilvitskii. Algorithms with predictions. In T. Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.

[30] M. Purohit, Z. Svitkina, and R. Kumar. Improving online algorithms via ML predictions. In *Proc. NeurIPS*, pages 9661–9670, 2018.

[31] D. Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proc. SODA*, pages 1834–1845, 2020.

[32] A. Wei and F. Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *Proc. NeurIPS*, 2020.

[33] A. Zeynali, B. Sun, M. Hajiesmaili, and A. Wierman. Data-driven competitive algorithms for online knapsack and set cover. In *Proc. AAAI*, pages 10833–10841, 2021.

Appendix (Omitted Tables)

	reserved types					
small type	$1b$	$1c$	$1d$	$1e$	$1e,4$	—
3	—	0.67	0.61	**0.56**	—	0.56
4	—	0.64	**0.58**	**0.53***	—	0.64
5	0.69	0.60	0.74	0.69	0.66	0.69
6	0.66	0.76	0.71	0.65	0.67	0.73
7	0.64	0.73	0.67	0.76	0.67	0.76
8	0.62	0.70	0.78	0.73	0.66	0.79
9	**0.61**	0.81	0.75	0.81	0.64	0.81
10	0.74	0.78	0.83	0.77	0.72	0.82
11	0.72	0.75	0.80	0.75	0.72	0.84
12	0.70	0.83	0.77	0.81	0.71	0.85
13	0.68	0.80	0.83	0.78	0.69	0.86
14	0.67	0.78	0.81	0.83	0.68	0.87
15	0.76	0.84	0.86	0.81	0.75	0.87
16	0.74	0.82	0.84	0.85	0.74	0.88
17	0.73	0.80	0.82	0.83	0.72	0.89
18	0.71	0.85	0.86	0.87	0.72	0.89
19	0.70	0.83	0.84	0.84	0.71	0.90
20	0.77	0.82	0.88	0.88	0.76	0.90
21	0.75	0.86	0.86	0.86	0.74	0.91
22	0.74	0.84	0.84	0.84	0.74	0.91
23	0.73	0.83	0.88	0.87	0.73	0.91
24	0.72	0.87	0.86	0.85	0.73	0.92
25	0.77	0.85	0.89	0.88	0.75	0.92
26	0.76	0.84	0.87	0.86	0.75	0.92
27	0.75	0.87	0.86	0.89	0.75	0.92
28	0.74	0.86	0.89	0.87	0.74	0.93
29	0.73	0.84	0.87	0.90	0.73	0.93
tiny	0.70	**0.61**	0.75	0.70	0.67	0.71

Table 2: A summary of the minimum occupied area in each LM bin type with no empty containers, rounded to 2 decimal places. For each large type, the minimum occupied area is highlighted. The entry marked with * shows the minimum occupied area of critical bins in the final packing.

Class	t_{i+1}	weight
$1a$	4/5	1.0
$1b$	2/3	0.765625
$1c$	3/5	0.6785570840932904
$1d$	11/20	0.4650876739816575
$1e$	1/2	0.4650876739816575
$2a$	2/5	0.25
$2b$	1/3	0.19140625
3	1/4	0.1111111111111111
4	1/5	0.0764160465740489
5	1/6	0.04
6	1/7	0.0277777777777777
7	1/8	0.0222880135840976
8	1/9	0.015625
9	1/10	0.0137867647058823
10	1/11	0.01
11	1/12	0.0082644628099173
12	1/13	0.0069444444444444
13	1/14	0.0059171597633136
14	1/15	0.0051020408163265
15	1/16	0.0044444444444444
16	1/17	0.00390625
17	1/18	0.0034602076124567
18	1/19	0.0030864197530864
19	1/20	0.002770083102493
20	1/21	0.0025
21	1/22	0.0022675736961451
22	1/23	0.0020661157024793
23	1/24	0.0018903591682419
24	1/25	0.0017361111111111
25	1/26	0.0016
26	1/27	0.0014792899408284
27	1/28	0.0013717421124828
28	1/29	0.0012755102040816
29	1/30	0.0011890606420927
tiny density		1.2500557840816486

Table 3: Weights of items of different classes, as used in Lemma 5

small types	1b containers	1b total area	1c containers	1c total area	1d containers	1d total area	1e containers	1e total area
	reserved types							
3	—	—	5	0.3125	5	0.3125	5	0.3125
4	—	—	7	0.2800	7	0.2800	7	0.2800
5	9	0.2500	9	0.2500	16	0.4444	16	0.4444
6	11	0.2244	20	0.4081	20	0.4081	20	0.4081
7	13	0.2031	24	0.3750	24	0.3750	33	0.5156
8	15	0.1851	28	0.3456	39	0.4814	39	0.4814
9	17	**0.1700**	45	0.4500	45	0.4500	56	0.5600
10	36	0.2975	51	0.4214	64	0.5289	64	0.5289
11	40	0.2777	57	0.3958	72	0.5000	72	0.5000
12	44	0.2603	80	0.4733	80	0.4733	95	0.5621
13	48	0.2448	88	0.4489	105	0.5357	105	0.5357
14	52	0.2311	96	0.4266	115	0.5111	132	0.5866
15	81	0.3164	125	0.4882	144	0.5625	144	0.5625
16	87	0.3010	135	0.4671	156	0.5397	175	0.6055
17	93	0.2870	145	0.4475	168	0.5185	189	0.5833
18	99	0.2742	180	0.4986	203	0.5623	224	0.6204
19	105	0.2625	192	0.4799	217	0.5424	240	0.5999
20	144	0.3265	204	0.4625	256	0.5804	279	0.6326
21	152	0.3140	245	0.5061	272	0.5619	297	0.6136
22	160	0.3024	259	0.4896	288	0.5444	315	0.5954
23	168	0.2916	273	0.4739	333	0.5781	360	0.6250
24	176	0.2816	320	0.5120	351	0.5615	380	0.6079
25	225	0.3328	336	0.4970	400	0.5917	429	0.6346
26	235	0.3223	352	0.4828	420	0.5761	451	0.6186
27	245	0.3125	405	0.5165	440	0.5612	504	0.6428
28	255	0.3032	423	0.5029	495	0.5885	528	0.6278
29	265	0.2944	441	0.4899	517	0.5744	585	0.6500
tiny	9	0.2571	9	0.2571	16	0.4571	16	0.4571

Table 4: Lower bounds for the area ccupied by tiny and small items in LM bins of Rap. The minimum occupied area over all classes is highlighted.

Minimum Ply Covering of Points with Unit Disks*

Stephane Durocher[†] J. Mark Keil[‡] Debajyoti Mondal[§]

Abstract

Let P be a set of points and let U be a set of unit disks in the Euclidean plane. A minimum ply cover of P with U is a subset of U that covers P and minimizes the number of disks that share a common intersection. The size of a minimum ply cover is called the minimum ply cover number. Biedl et al. [Comput. Geom., 94:101712, 2020] showed that determining the minimum ply cover number for a set of points by a set of unit disks is NP-hard, and asked whether there exists a polynomial-time $O(1)$-approximation algorithm for this problem. They showed the problem to be 2-approximable in polynomial time for the special case when the minimum ply cover number is constant. In this paper, we settle the question posed by Biedl et al. by providing a polynomial-time $O(1)$-approximation algorithm for the minimum ply cover problem.

1 Introduction

The *minimum set cover problem* is a widely studied optimization problem. The input to the set cover problem is a set P and a collection C of subsets over P. The goal is to identify a subset C' of C with minimum cardinality that contains all the elements of P. The *membership* of an element q in P with respect to a subset C' of C is the number of sets in C' that contain q. The *minimum membership set cover problem* is a variant in which the goal is to find a subset C' of C that minimizes the maximum membership of elements in P. A rich body of literature studies the minimum membership set cover problem [2, 10, 12, 13, 15, 16]. In this paper, we consider a set cover scenario in which the given sets of C may contain elements outside P and membership is evaluated for all elements covered by C', including those outside P. This concept appears in the literature as *ply cover*, which is formalized below.

The *ply* of a collection S of sets, denoted $\text{ply}(S)$, is the maximum cardinality of any subset of S that has a non-empty common intersection. The set S *covers* a set P if $P \subseteq \bigcup_{S_i \in S} S_i$. Given a set P and a collection of sets U, a subset $S \subseteq U$ is a *minimum ply cover* of P if S covers P and S minimizes $\text{ply}(S)$ over all subsets of U. Formally:

$$\text{plycover}(P, U) = \underset{\substack{S \subseteq U \\ S \text{ covers } P}}{\arg\min}\ \text{ply}(S). \quad (1)$$

The ply of such a set S is called the *minimum ply cover number* of P with U, denoted $\text{ply}^*(P, U)$. For example, if $P = \{1, 3, 5, 7, 8\}$ and $U = \{\{1, 2, 3, 4\}, \{8\}, \{3, 4, 5\}, \{4, 5, 7\}\}$, then $\text{plycover}(P, U) = \{\{1, 2, 3, 4\}, \{8\}, \{4, 5, 7\}\}$ and the minimum ply cover number is two.

Motivated by applications in covering problems, including interference minimization in wireless networks, Biedl et al. [3] introduced the *minimum ply cover problem* in the geometric setting: given sets P and U, find a subset $S \subseteq U$ that minimizes (1). When U is a set of unit disks representing transmission ranges of potential locations for placing wireless transmitters and P represents locations of wireless clients, $S \subseteq U$ corresponds to locations to install transmitters that minimize interference at any point in the plane.

Biedl et al. [3] showed that the problem is NP-hard to solve exactly, and remains NP-hard to approximate by a ratio less than two when P is a set of points in $\mathbb{R}^2$ and U is a set of axis-aligned unit squares or a set of unit disks in $\mathbb{R}^2$. They also provided 2-approximation algorithms parameterized in terms of $\text{ply}^*(P, U)$ for unit disks and unit squares in $\mathbb{R}^2$. Their algorithm for axis-parallel unit squares runs in $O((k + |P|)(2 \cdot |U|)^{3k+1})$ time, where $k = \text{ply}^*(P, U)$, which is polynomial when $\text{ply}^*(P, U) \in O(1)$.

Biniaz and Lin [4] generalized this result for any fixed-size convex shape and obtained a 2-approximation algorithm when $\text{ply}^*(P, U) \in O(1)$. The problem of finding a polynomial-time approximation algorithm to the minimum ply cover problem remained open for both unit squares and unit disks when the minimum ply cover number, $\text{ply}^*(P, U)$, is not bounded by any constant.

Recently, Durocher et al. [11] settled this question affirmatively for unit squares by designing a polynomial-time $(8 + \varepsilon)$-approximation algorithm for the problem, where $\varepsilon > 0$. We refer the reader to [19] for subsequent work that achieves faster algorithms, but with larger approximation factors.

Our contribution: In this paper we consider the minimum ply cover problem for a set P of points in $\mathbb{R}^2$

*This work is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[†]University of Manitoba, Winnipeg, Canada, stephane.durocher@umanitoba.ca

[‡]University of Saskatchewan, Saskatoon, Canada, keil@cs.usask.ca

[§]University of Saskatchewan, Saskatoon, Canada, dmondal@cs.usask.ca

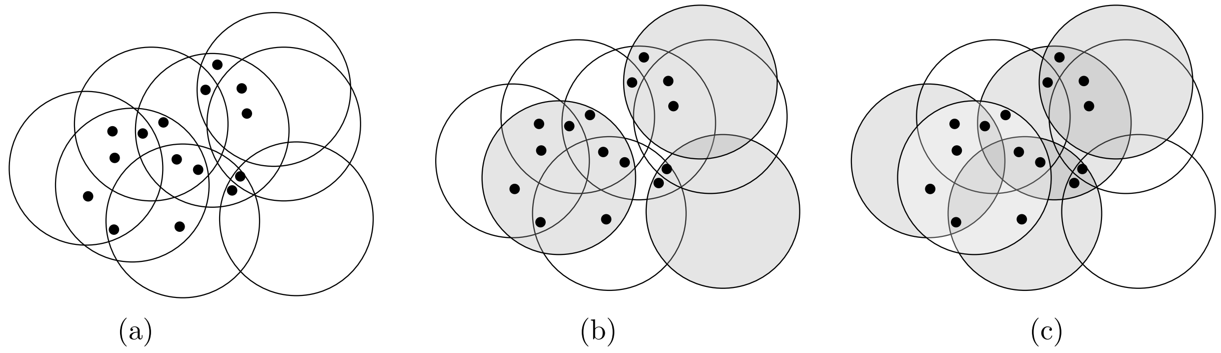


Figure 1: (a) An input consisting of points and unit disks. (b) A covering of the points with ply 1, which is also the minimum ply cover number for the given input. (c) A covering of the same instance with ply 3.

with a set U of unit disks in $\mathbb{R}^2$. We show that for every $\varepsilon > 0$, the minimum ply cover number can be approximated in polynomial time for unit disks within a factor of $(63+\varepsilon)$. This settles an open question posed in [3] and [4].

Our idea is to leverage the *minimum discrete unit disk cover problem* that seeks to cover a given point set with a smallest cardinality subset of the given disks. We show that there exist instances where the cardinality of the minimum discrete unit disk cover is at least 9.24 times the minimum ply cover. Hence, obtaining an approximation factor of 10 would be interesting, and we believe that achieving an approximation factor smaller than 10 would require a different technique that does not rely predominantly on a discrete unit disk cover.

Recent Developments: Recently, and independently of our work, Bandyapadhyay et al. [1] have shown that minimum ply cover can be approximated within a constant factor in $O(n\cdot\text{polylog}(n))$ time for fat objects, which includes unit disks and unit squares. Their idea is similar to the one that we used for disks. For unit squares, the technique yields an approximation factor of 36. For disks, they only provide a high-level argument for obtaining an $O(1)$-factor approximation rather than aiming for an exact value.

2 Approximating Minimum Ply Covering by Discrete Unit Disk Cover

Let P be a set of points in $\mathbb{R}^2$ and let U be a set of unit disks in $\mathbb{R}^2$. We assume that no three disks in U have boundaries that intersect at a common point. In this section we give a polynomial-time algorithm to approximate the minimum ply cover number for P with U within a factor of $O(1)$. We first give an overview of the algorithm and then describe its details.

2.1 Overview

Consider an axis-aligned grid $\mathcal{G}$ over P, where each grid cell is of size $(1/\sqrt{2}) \times (1/\sqrt{2})$. We choose a grid that is in general position relative to the disks in U, i.e., no disk is tangent to a grid line. A grid cell is called *non-empty* if it contains some point of P, otherwise, we call it *empty*.

We leverage the *minimum discrete unit disk cover problem* that, given a set of points and a set of unit disks on the Euclidean plane, seeks a minimum-cardinality subset of the input disks that covers the input points, for which a PTAS exists [17]. We show that one can first find an approximate solution to the minimum discrete unit disk cover for each non-empty grid cell, and then combine the solutions to obtain an approximate solution to the minimum ply cover for P.

2.2 Details of the Algorithm

We first remove all the disks in U that do not contain any point of P as they are not needed for covering P. Let R be a non-empty grid cell of $\mathcal{G}$. We first provide an upper bound on the cardinality of the minimum discrete unit disk cover in terms of the minimum ply cover number for the points and disks that overlap R (Lemma 1). We then show how to combine the respective solutions from each cell to obtain a cover of P by a subset of U whose ply cover number is at most $(63+\varepsilon)\,\text{ply}^*(P, U)$ (Theorem 2).

Lemma 1 *Let $Q \subseteq P$ be the points that lie in R and let $W \subseteq U$ be the set of unit disks that intersect R. Let S be a set of k points in the plane (i.e., not necessarily in P) such that every disk in W includes at least one point in S (points in S may lie outside R). The cardinality of every minimum discrete unit disk cover of Q by W is at most k times the minimum ply cover number for Q.*

Proof. Let δ be the cardinality of a minimum discrete unit disk cover for covering Q by W. Let β be the

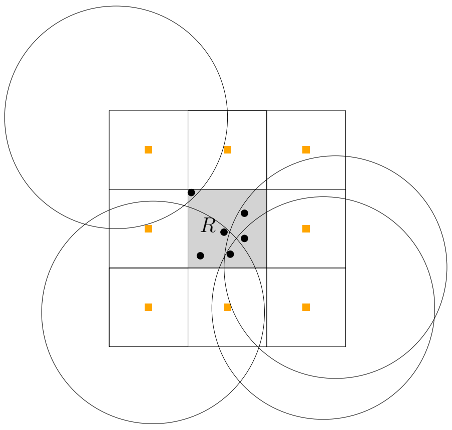


Figure 2: Illustration for Corollary 1.1, where R is shown in gray, Q is shown in black disks and S is shown in orange. Any disk that intersects the center grid cell must cover at least one orange point.

minimum ply cover number for covering Q by W. If $\delta \leq k\beta$, then $\beta \geq \delta/k$.

Suppose for a contradiction that the minimum ply cover number is less than δ/k. Since every disk in the minimum ply cover must hit at least one point in S, the number of disks in the cover is strictly less than δ. This contradicts our initial assumption that δ is the cardinality of a minimum discrete unit disk cover of Q. □

It is straightforward to verify that for Lemma 1, it suffices to choose the centers of the 8 neighbouring cells of R as the point set S (Figure 2). Specifically, let D be a unit disk that intersects R. The unit disks centered at the points of S cover the entire region inside the convex hull of S. Therefore, if the center of D lies inside the convex hull of S, then D must include at least one point from S. The remaining case is when the center of D lies outside of S. If D does not include the points of S, then it can intersect a segment of length at most $1/\sqrt{2}$ from the convex hull boundary of S. However, this chord length is too short for D to reach R, which contradicts the assumption that D intersects R. Hence we obtain the following corollary.

Corollary 1.1 *Let $Q \subseteq P$ be the points that lie in R and let $W \subseteq U$ be set of unit disks that intersect R. The cardinality of a minimum discrete unit disk cover for Q by W is at most 8 times the minimum ply cover number for Q by W.*

In the following theorem we show how to combine the approximate solutions for the cells of $\mathcal{G}$ to obtain an $O(1)$-approximation for the minimum ply cover problem.

Theorem 2 *Let P be a set of points and let U be a set of unit disks, both in $\mathbb{R}^2$. Assume that for every $Q \subseteq P$ and $W \subseteq U$, there exists a $f(Q, W)$-time algorithm $\mathcal{A}$ that can approximate the cardinality of the minimum discrete unit disk cover of Q with W within a factor of γ. Then the minimum ply cover number for P using U can be approximated within a factor of 360γ in $O(|P| \cdot f(P, U))$ time.*

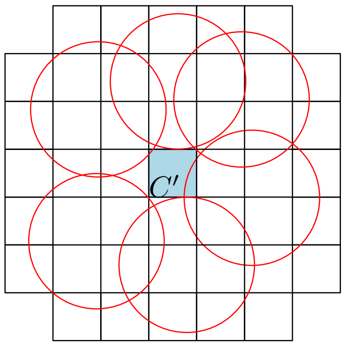


Figure 3: The friend cells for C'. The red circles illustrate that for every friend cell, there is a unit disk that intersects both that cell and C'.

Proof. Let U^* be a minimum ply cover for covering P with U. We consider a grid $\mathcal{G}$ over the point set P where each grid cell is of size $(1/\sqrt{2}) \times (1/\sqrt{2})$. Apply the algorithm $\mathcal{A}$ iteratively to find a γ-approximation for the cardinality of the minimum discrete unit disk cover for each grid cell. Let the maximum cardinality that we attain for a cell be δ_{max}. Let C be the cell that attains δ_{max}, and let Q_C and W_C be the points and unit disks corresponding to C, respectively. By Corollary 1.1, the cardinality of the minimum discrete unit disc cover is at most 8 times the minimum ply cover number for covering Q_C with W_C. Therefore, δ_{max} at most 8γ times the minimum ply cover number for Q_C. Since $Q_C \subseteq P$ and $W_C \subseteq U$, the minimum ply cover number for covering Q_C with W_C is smaller than the minimum ply cover number ($\text{ply}(U^*)$) for covering P with U. Therefore, we have $\delta_{max} \leq 8\gamma \,\text{ply}(U^*)$.

Let $\mathcal{O}$ be the union of all the approximate discrete unit disk covers obtained by applying the algorithm $\mathcal{A}$ to cells of $\mathcal{G}$, and let r be a point in the plane that does not fall on any grid line of $\mathcal{G}$. Let C' be the cell of $\mathcal{G}$ that contains r. In the following we show that r can belong to at most $45\delta_{max}$ disks in $\mathcal{O}$.

We refer to a cell D to be a *friend* of C' if a solution to the discrete unit disk cover for covering Q_D intersects C'. In other words, for every friend D, there is a unit disk that intersects both D and C'. There are 45 friend cells for C' (see Figure 3). Therefore, the number of disks that contains r in $\mathcal{O}$ is at most $45\delta_{max}$. Since $\delta_{max} \leq 8\gamma \,\text{ply}(U^*)$, the number of unit disks in $\mathcal{O}$ that may contain r is at most $360\gamma \,\text{ply}(U^*)$. Thus the ply of $\mathcal{O}$ is at most $360\gamma \,\text{ply}(U^*)$. □

Since there exists a PTAS for the discrete unit disk cover problem [17], we obtain the following corollary.

Corollary 2.1 *Given a set P of points and a set U of unit disks, both in $\mathbb{R}^2$, a ply cover of P using U can be computed in polynomial time whose ply is within a constant factor of the minimum ply cover number of P by U.*

3 Further Improvements

Note that we have some freedom when choosing the set S in Lemma 1 and the grid resolution in Theorem 2. Therefore, it is natural to leverage such freedom to further lower the approximation factor.

Note that there are several choices for S. For example, consider a regular pentagon inscribed in a unit circle centered at the center of R. Once can choose the corners of the pentagon as the points of S, as illustrated in Figure 4. Specifically, every unit disk with center lying inside the unit circle (shown in red) includes at least one point from S, and every unit disk with center lying outside the unit circle and avoiding S is unable to reach R, as illustrated in blue disks.

If we choose the corners of the pentagon as the points of S, then the approximation factor 8 in Corollary 1.1 improves to 5 and the overall approximation factor in Theorem 2 improves to $45 \cdot 5 \cdot \gamma = 225\gamma$. The factor 225 is determined partly by the number of fried cells, which is 45. To reduce this factor, we choose a hexagonal grid instead of a square grid. This requires us to design a new set of S, but it turns out that the overall approximation factor reduces to 63γ. We now give the details of the construction.

Let H be a regular hexagon that inscribes a unit circle with a side parallel to the x-axis (Figure 5(a)). Consider now a hexagonal grid $\mathcal{H}$ on the point set P where each hexagon is a copy of H. We compute the approximate discrete unit disk cover for each cell of H. Let δ_{max} be the largest approximate discrete unit disk cover that has appeared for a cell C.

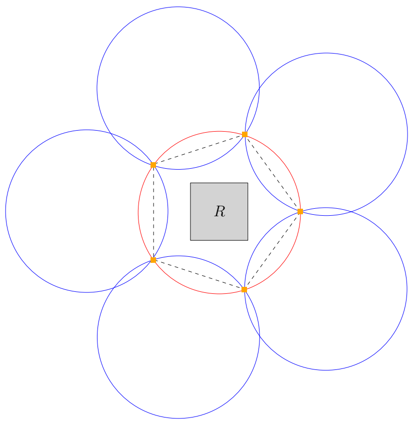


Figure 4: An alternative choice for S.

Observe that each hexagonal cell can be partitioned into 6 triangles by drawing a line segment between opposite corners of the hexagon (Figure 5(b)). While combining the solutions, we consider each triangular region instead of each hexagonal region, as follows.

Let T be a triangular region, as illustrated in Figure 5(c). We first use the idea of Lemma 1 to compute an upper bound on the minimum discrete unit disk cover for the points and unit disks corresponding to T. To obtain such an upper bound, we design a set S of 7 points such that any unit disk intersecting T contains at least one point from S. Let H' be the hexagonal cell that contains T and let o be the center of T. Then S includes the point o and the 6 points obtained from the intersection of the hexagonal grid and the circle of radius 1.5 centered at o. Figure 5(c) illustrates the circle of radius 1.5 in dashed lines and the points of S in orange. To verify that any unit disk D that intersects T contains a point from S, consider two cases. If the center c of D lies inside the hexagon H'' determined by $S \setminus \{o\}$, then c lies in an equilateral triangle with side length 1.5, which is determined by three points of S. Figure 5(c) illustrates the equilateral triangle in green. The radius of the circumscribed circle of this equilateral triangle is $1.5/\sqrt{3} < 1$. Therefore, D must contain a point from S. If the center c of D lies outside H'', then it can reach T only when D passes through two points of S, as illustrated in Figure 5(d).

We now compute the approximation factor using the same proof technique as in Theorem 2. Let $\mathcal{O}$ be the union of all the approximate discrete unit disk covers obtained by applying the algorithm $\mathcal{A}$ to the hexagonal cells of $\mathcal{H}$, and let r be a point in the plane that does not fall on any grid line of $\mathcal{H}$. Let C' be a triangular region that contains r. We now count the hexagonal cells that are within unit distance to the C'. In other words, the discrete unit disk cover solution for only these cells may contain r. There are 9 friend cells for C' (see Figure 5(e)). Therefore, the number of disks that contains r in $\mathcal{O}$ is at most $9\delta_{max}$. Since $|S| = 7$, we have $\delta_{max} \leq 7\gamma\,\text{ply}(U^*)$, where γ is the approximation factor for the minimum discrete unit disk cover and U^* is the minimum ply cover. Consequently, the number of unit disks in $\mathcal{O}$ that may contain r is at most $9 \cdot 7 \cdot \gamma\,\text{ply}(U^*)$. Thus the ply of $\mathcal{O}$ is at most $63\gamma\,\text{ply}(U^*)$. Since there is a polynomial-time $(1+\varepsilon')$-approximation for the minimum discrete unit disk cover [17], we obtain a $(63 + \varepsilon)$-approximation for the minimum ply cover number where we choose ε' to be $\varepsilon/63$.

The following theorem summarizes the result of this section.

Theorem 3 *Given a set P of points and a set U of unit disks, both in $\mathbb{R}^2$, and a constant $\varepsilon > 0$, a ply cover of P*

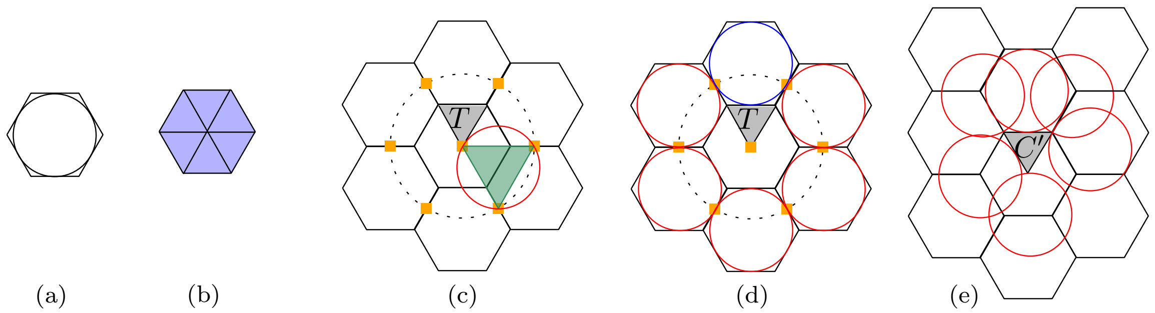


Figure 5: Improving the approximation factor by choosing a hexagonal grid.

using U can be computed in polynomial time whose ply is at most $(63+\varepsilon)$ times the minimum ply cover number of P by U.

The bottleneck of the running time of our algorithm is the time to compute the discrete unit disk cover. In 1995, Brönnimann and Goodrich gave an $O(1)$-approximation algorithm for minimum discrete unit disk cover [5]. A rich body of research attempted to lower the approximation factor since then [6, 18, 7, 8]. The $(1+\varepsilon)$-approximation result for the minimum discrete unit disk cover [17] has a running time of $O(m^{2(c/\varepsilon)^2+1}n)$, where m and n are the numbers of disks and points, respectively, and c is a constant. This running time is large, i.e., the fastest achievable running time is $O(m^{65}n)$ by setting $\varepsilon = 2$, which gives a 3-approximation [14]. Das et al. [9] gave an 18-approximation algorithm that runs in $O(n \log n + m \log m + mn)$ time, which may be used to compute an approximate solution to the minimum ply cover problem faster, but the approximation factor would increase to 1134.

4 Lower Bound

Our approximation algorithm for the minimum ply cover problem relies heavily on finding a discrete unit disk cover. In this section, we construct instances where the cardinality of the minimum discrete unit disk cover is at least 9.2444 times the minimum ply cover number. The bound 9.24 is constructed to complement our approach, i.e., in general, the number of disks in a discrete unit disk cover could be unbounded compared to the minimum ply cover number. This 9.24 lower bound indicates that achieving an approximation factor less than 10 may be unlikely using our approach.

Choose any $n \geq 2$. We construct a set $\{D_1, \ldots, D_n\}$ of n unit disks such that the boundary of each disk is tangent to a common point o (each disk center is a unit distance from o), and the disks are positioned uniformly around o. Figure 6 shows these disks in gray. Consider a circle C of radius 2 centered at o (shown in orange in Figure 6). For each $i \in \{1, \ldots, n\}$, we add a point p_i (shown in red in Figure 6) at the intersection of the boundaries of C and D_i, and place a unit disk D'_i (shown in black in Figure 6) such that p_i is the midpoint of the centers of D_i and D'_i.

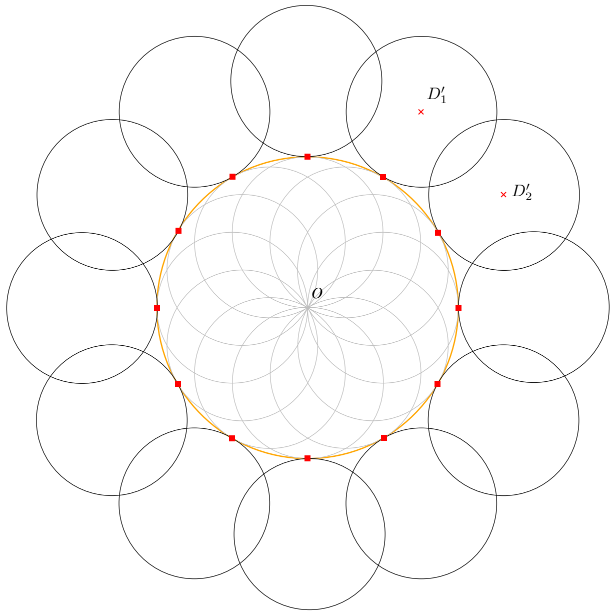


Figure 6: Illustration for the construction of a ply cover instance (P, U) when $n = 12$. The points of P are shown in red, and U consists of the black and gray disks.

Consider an instance of the minimum ply cover problem (P, U), where $P = \{p_1, \ldots, p_n\}$ and $U = \{D_1, \ldots, D_n, D'_1, \ldots, D'_n\}$. For each $i \in \{1, \ldots, n\}$, the point $p_i \in P$ is covered by exactly two disks in U, D_i and D'_i; furthermore, D_i and D'_i cover no points in $P \setminus \{p_i\}$. Therefore, any disk cover of P by U must contain at least n disks and must contain either D_i or D'_i for each $i \in \{1, \ldots, n\}$.

The set $U' = \{D'_1, \ldots, D'_n\}$ covers P and $|U'| = n$. Therefore, U' is a minimum discrete unit disk cover of P. Similarly, the set $U'' = \{D_1, \ldots, D_n\}$ covers P, $|U''| = n$, and U'' is also a minimum discrete unit disk cover of P. U'' has ply n. We now calculate the ply of U'.

See Figure 7, illustrating the point o and disks D_i and D'_i, for some $i \in \{1, \ldots, n\}$. The segment $\overline{oc_i}$ is the diameter of D_i plus the radius of D'_i; therefore it has length 3. Consequently, $\theta = 2\sin^{-1}(1/3)$, and the ply of U' is $\lceil \frac{n 2\sin^{-1}(1/3)}{2\pi} \rceil$.

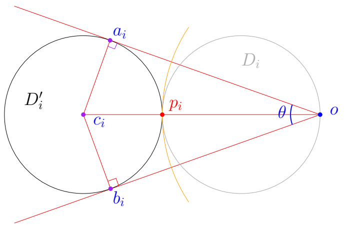


Figure 7: The sector rooted at o with boundary tangent to the disk D'_i forms an angle $\theta = 2\sin^{-1}(1/3)$ at o.

An adversarial choice of minimum discrete unit disk cover of P by U selects U'. Consequently, no minimum discrete unit disk cover can guarantee to approximate the minimum ply by less than

$$\begin{aligned}\lim_{n\to\infty} \frac{\mathrm{ply}(U'')}{\mathrm{ply}(U')} &= \lim_{n\to\infty} \frac{n}{\lceil 2n\sin^{-1}(1/3)\rceil/(2\pi)} \\ &= \frac{\pi}{\sin^{-1}(1/3)} \\ &> 9.2444.\end{aligned}$$

The following theorem summarizes the result of this section.

Theorem 4 *For sufficiently large n, there exists a set of n points and $2n$ disks for which the ply of a minimum discrete unit disk cover is at least 9.24 times the minimum ply cover.*

5 Conclusion

We have shown that given a set of points and a set of unit disks in the Euclidean plane, one can compute a ply cover whose ply is within a constant factor of the minimum ply cover number. The approximation factor we obtain is large (i.e., $63 + \varepsilon$), whereas only a 2-inapproximability result is known [3]. Therefore, a natural direction of future research is to narrow down this gap.

Our approximation algorithm relies on finding an approximate discrete unit disk cover and we have constructed instances where a minimum discrete unit disk cover is at least 9.24 times the minimum ply cover number. This raises the question of whether the approximation factor could be brought down closer to 10, or whether the existing 2-inapproximability result could be strengthened further using the disk configurations that we used in this paper.

References

[1] S. Bandyapadhyay, W. Lochet, S. Saurabh, and J. Xue. Minimum-membership geometric set cover, revisited. In *Proceedings of the 39th International Symposium on Computational Geometry (SoCG 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[2] M. Basappa and G. K. Das. Discrete unit square cover problem. *Discret. Math. Algorithms Appl.*, 10(6):1850072:1–1850072:18, 2018.

[3] T. C. Biedl, A. Biniaz, and A. Lubiw. Minimum ply covering of points with disks and squares. *Comput. Geom.*, 94:101712, 2021.

[4] A. Biniaz and Z. Lin. Minimum ply covering of points with convex shapes. In *Proc. 32nd Canadian Conference on Computational Geometry (CCCG)*, pages 2–5, 2020.

[5] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discret. Comput. Geom.*, 14(4):463–479, 1995.

[6] G. Călinescu, I. Măndoiu, P.-J. Wan, and A. Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. In *Proceedings of the 5th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 34–43, 2001.

[7] P. Carmi, M. J. Katz, and N. Lev-Tov. Covering points by unit disks of fixed location. In *Algorithms and Computation: 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007. Proceedings 18*, pages 644–655. Springer, 2007.

[8] F. Claude, G. K. Das, R. Dorrigiv, S. Durocher, R. Fraser, A. López-Ortiz, B. G. Nickerson, and A. Salinger. An improved line-separable algorithm for discrete unit disk cover. *Discret. Math. Algorithms Appl.*, 2(1):77–88, 2010.

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

[10] E. D. Demaine, U. Feige, M. Hajiaghayi, and M. R. Salavatipour. Combination can be hard: Approximability of the unique coverage problem. *SIAM J. Comput.*, 38(4):1464–1483, 2008.

[11] S. Durocher, J. M. Keil, and D. Mondal. Minimum ply covering of points with unit squares. In *Proc. of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM)*, volume 13973 of *LNCS*, pages 23–35. Springer, 2023.

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

[13] T. Erlebach and E. J. van Leeuwen. PTAS for weighted set cover on unit squares. In M. J. Serna, R. Shaltiel, K. Jansen, and J. D. P. Rolim, editors, *Proc. of the 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX)*, volume 6302 of *LNCS*, pages 166–177. Springer, 2010.

[14] R. Fraser and A. López-Ortiz. The within-strip discrete unit disk cover problem. *Theor. Comput. Sci.*, 674:99–115, 2017.

[15] F. Kuhn, P. Rickenbach, R. Wattenhofer, E. Welzl, and A. Zollinger. Interference in cellular networks: The minimum membership set cover problem. In *Proc. of the 11th Conference on Computing and Combinatorics (COCOON)*, volume 3595 of *LNCS*, pages 188–198. Springer-Verlag, 2005.

[16] N. Misra, H. Moser, V. Raman, S. Saurabh, and S. Sikdar. The parameterized complexity of unique coverage and its variants. *Algorithmica*, 65(3):517–544, 2013.

[17] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discret. Comput. Geom.*, 44(4):883–895, 2010.

[18] S. Narayanappa and P. Vojtechovský. An improved approximation factor for the unit disk covering problem. In *Proceedings of the 18th Annual Canadian Conference on Computational Geometry, CCCG 2006, August 14-16, 2006, Queen's University, Ontario, Canada*, 2006.

[19] S. Sarkar. Faster algorithm for minimum ply covering of points with unit squares. *arXiv preprint arXiv:2301.13108*, 2023.

Overlapping of Lattice Unfolding for Cuboids

Takumi Shiota* Tonan Kamata† Ryuhei Uehara†

Abstract

A polygon obtained by cutting the surface of a polyhedron is called an unfolding of the polyhedron. An unfolding obtained by cutting along only edges is called an edge unfolding. An unfolding may have overlapping, which are self-intersections on its boundary. It is a famous open problem in computational origami whether or not every convex polyhedron has a non-overlapping edge unfolding. Recently, to get a foothold on the open problem, an overlapping of unfolding by other cutting restrictions was studied. Lattice unfoldings of a cuboid made by unit cubes are a specific example. A lattice unfolding of a cuboid is a polygon obtained by cutting the faces along the edges of unit squares. An unfolding may have overlapping, even in the case of small cuboids. In particular, Uno showed that a $(1, 1, 3)$-cuboid has an overlapping lattice unfolding, while Mitani and Uehara showed the same for three faces of a $(1, 2, 3)$-cuboid. In contrast, it is known that some cuboids have no overlapping lattice unfolding. Hearn showed it for a $(1, 1, 2)$-cuboid, and Sugihara showed the same for a $(2, 2, 2)$-cuboid. In this study, we completely determine the existence of overlapping lattice unfoldings which also contains the case where the sizes are non-integers.

1 Introduction

To represent a polyhedron, we sometimes use a planer layout of arranged faces according to their adjacency relations. The origin of this method can be traced back to Albrecht Dürer's 1525 book "Underweysung der messung mit dem zirckel un richt scheyt" [4]. He represented several polyhedra using flat polygons (edge unfoldings) obtained by cutting along the edges. All edge unfoldings of convex polyhedra in this book are drawn so that "no two faces overlap or in touch." However, edge unfoldings of polyhedra do not always satisfy this condition (e.g., Namiki and Fukuda's overlapping edge unfolding as shown in Figure 1 [11]). The following problem is open:

Open Problem 1 ([5], Open Problem 21.1) *Does every convex polyhedron have a non-overlapping edge unfolding?*

*Kyushu Institute of Technology, shiota.takumi779@mail.kyutech.jp

†Japan Advanced Institute of Science and Technology, {kamata,uehara}@jaist.ac.jp

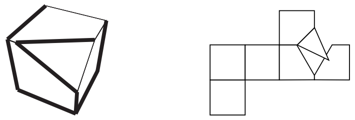


Figure 1: An overlapping edge unfolding of a cube with cut-off corners [11]. Cut along thick lines to get the figure on the right.

Table 1: Overlapping edge unfoldings for convex regular-faced polyhedra

Convex regular-faced polyhedra	Is there an overlapping edge unfolding?
Platonic solids (Total 5 types)	No [9]
Archimedean solids (Total 13 types)	No (7 types) [8, 14] Yes (6 types) [3, 9, 14]
n-gonal Archimedean prisms ($n \geq 3$)	No ($3 \leq n \leq 23$) Yes ($n \geq 24$) [14]
n-gonal Archimedean antiprisms ($n \geq 3$)	No ($3 \leq n \leq 11$) Yes ($n \geq 12$) [14]
Johnson solids (Total 92 types)	No (48 types) Yes (44 types) [13]

Research on the existence of unfolding with overlap for polyhedra has been conducted under several different conditions. Biedl et al. discovered concave polyhedra where all edge unfoldings overlap in 1998, and Grünbaum found another instance in 2003 [2, 6]. For the class of polyhedra whose faces are all regular polygons, referred to as convex regular-faced polyhedra, it has been completely determined whether they have overlapping edge unfoldings (see Table 1).

There are also studies on general unfoldings that allow cutting the faces of the polyhedron, not just its edges. Sharir and Schorr showed that any convex polyhedron could unfold without overlapping when allowed to cut its faces [12, 1].

In our study, we consider another restriction of unfolding called lattice unfolding, which can be applied to cuboids of specific sizes.

2 Preliminaries

2.1 Definition of cuboids

We consider a square lattice where each square has an area of 1×1. Let A and B be a pair of lattice points and a and b be the differences in x coordinates and y

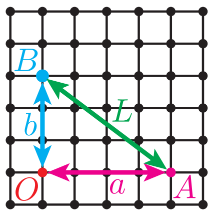
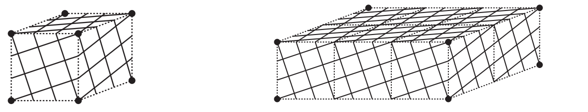
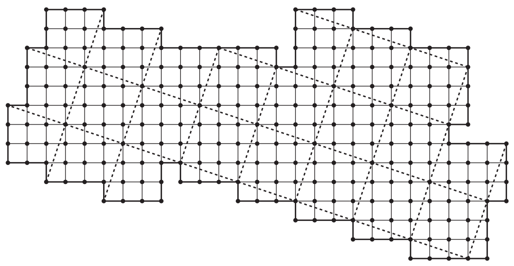


Figure 2: Determining the length L of one edge of a cube.



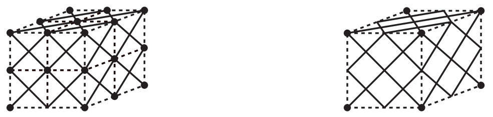
(a) The cube with length $\sqrt{10}$ on a side.

(b) A $(3\sqrt{10}, 2\sqrt{10}, 1\sqrt{10})$-cuboid obtained by connecting six units of (a).



(c) An example of the lattice unfolding of (b).

Figure 3: Examples of cube, cuboid, and lattice unfolding



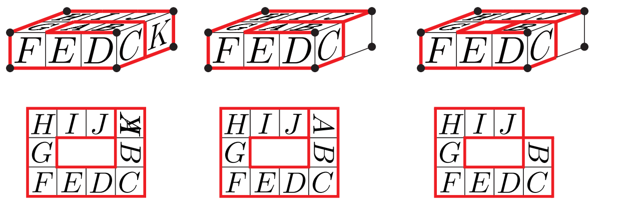
(a) A $(2\sqrt{2} \times 2\sqrt{2} \times 2\sqrt{2})$-cuboid ($L = \sqrt{2}$, $x = y = z = 2$).

(b) A $(2\sqrt{2} \times 2\sqrt{2} \times 2\sqrt{2})$-cuboid ($L = 2\sqrt{2}$, $x = y = z = 1$).

Figure 4: Two cuboids which can be regarded as the same shape.

coordinates between them (Figure 2). We assume $a \geq b$ without loss of generality. Here, we consider a square with a side AB, whose length is $L = \sqrt{a^2 + b^2}$. We construct a *cube with length L on a side* by assembling the squares as its faces (Figure 3(a)).

We also define a *(xL, yL, zL)-cuboid* by a box with edge lengths xL, yL, and zL along x-axis, y-axis, and z-axis, respectively for some positive integers x, y, and z (Figure 3(b)). We assume $x \leq y \leq z$ without loss of generality. We only consider the cuboids that satisfy $\gcd(a, b) = 1$ because the $(c(xL), c(yL), c(zL))$-cuboid (multiplied (xL, yL, zL)-cuboid by c) and the $(x(cL), y(cL), z(cL))$-cuboid (multiplied (cL, cL, cL)-cuboid by x, y, z) can be regarded as the same shape (Figure 4).



(a) Faces-in-touch **(b)** Edges-in-touch **(c)** Vertices-in-touch

Figure 5: Overlapping lattice unfoldings in the (1, 2, 3)-cuboid [10].

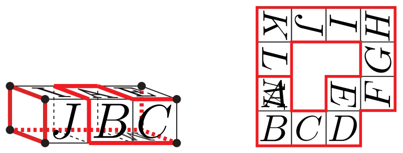


Figure 6: A faces-in-touch unfolding in the (1, 1, 3)-cuboid [16].

2.2 Definition of overlapping lattice unfoldings

A *lattice unfolding* of a cuboid is an unfolding obtained by cutting the face of the cuboid along the edges of unit squares (Figure 3(c)). As we will mention in Lemma 7, the cutting line of the lattice unfolding forms a tree structure.

On a lattice unfolding, the original cuboid's unit squares are arranged planarly so that their edges are glued together. Any pair of unit squares not adjacent to each other on the surface can be classified into positional relationships as follows:

(1) Overlap in the same position (Figure 5(a)).
(2) Share one edge (Figure 5(b)).
(3) Share one vertex (Figure 5(c)).
(4) Without sharing any edges or vertices.

Herein, we say that an unfolding is *faces-in-touch* if it has a pair of unit squares satisfying (1). Similarly, we define *edges-in-touch* and *vertices-in-touch* for (2) and (3), respectively. When all pairs of unit squares not adjacent on the surface satisfy (4), it is called *non-overlapping*. When any of the conditions (1), (2), or (3) are satisfied, it is termed *overlapping*. Note that the inclusion relationship {faces-in-touch unfoldings}⊂{edges-in-touch unfoldings}⊂{vertices-in-touch unfoldings} holds for any cuboid.

2.3 Background on overlapping lattice unfoldings

The overlapping of lattice unfoldings has been mainly researched in the case of $L = 1$, which is the size of unit cubes.

In 2008, Uno showed that the (1, 1, 3)-cuboid has a faces-in-touch lattice unfolding (Figure 6) [16]. Furthermore, in 2008, Mitani and Uehara showed that the $(1, 2, 3)$-cuboid has a faces-in-touch lattice unfolding (Figure 5(a)) [10].

Each of these cutting methods can be extended to the cases of $(1, 1, z)$, where $z \geq 3$ and $(1, y, z)$, where $y \geq 2, z \geq 3$, respectively, and the following theorems are obtained:

Theorem 2 ([16]) *A faces-in-touch lattice unfolding exists for any $(1, 1, z)$-cuboid, where $z \in \mathbb{N}, z \geq 3$.*

Theorem 3 ([10]) *A faces-in-touch lattice unfolding exists for any $(1, y, z)$-cuboid, where $y, z \in \mathbb{N}, y \geq 2, z \geq 3$.*

On the other hand, the following results are known for the non-existence of overlapping unfolding:

Theorem 4 ([7]) *A faces-in-touch lattice unfolding does not exist for the $(1, 1, 2)$-cuboid.*

Theorem 5 ([15]) *A faces-in-touch lattice unfolding does not exist for the $(2, 2, 2)$-cuboid.*

We have revisited the classes described above and compiled an extended list that applies to a wider range of classes.

2.4 Representation of polyhedra using graphs

Let Q be a polyhedron, and let $G_Q = (V_Q, E_Q)$ be the graph such that V_Q is the set of the vertices of Q and E_Q is the set of the edges of Q. We call this graph an *edge representation graph* of Q. An edge unfolding of Q can be regarded as an unfolding obtained from a subgraph of G_Q. The following lemma holds:

Lemma 6 (See e.g., [5], Lemma 22.1.1) *A subgraph $G \subset G_Q$ yields an unfolding if and only if G is a spanning tree of G_Q.*

Now, we introduce a new graph representation for lattice unfoldings. Let C be a cuboid. We define the *lattice representation graph* $G_C = (V_C, E_C)$ such that V_C is the set of vertices of unit squares on the face of C, and E_C is the set of edges of the unit squares. The lattice unfolding is one of the general unfolding, which allows cutting the surface across faces. Thus, we can apply the following lemma, which holds for the general unfolding (see Figure 7):

Lemma 7 ([10], Theorem 1, Theorem 3) *Let $G_C = (V_C, E_C)$ be the lattice representation graph of a cuboid C, and let $S(V_C) \subset V_C$ be the set of lattice points located at the vertices of C. Then, the following are equivalent for a subgraph $G \subset G_C$:*

1. *G yields a lattice unfolding.*
2. *G is a tree that satisfies $S(V_C) \subset G$, and for any vertex v in G, if $deg(v) = 1$, then $v \in S(V_C)$ (where $deg(v)$ is the degree of vertex v).*

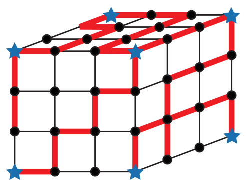
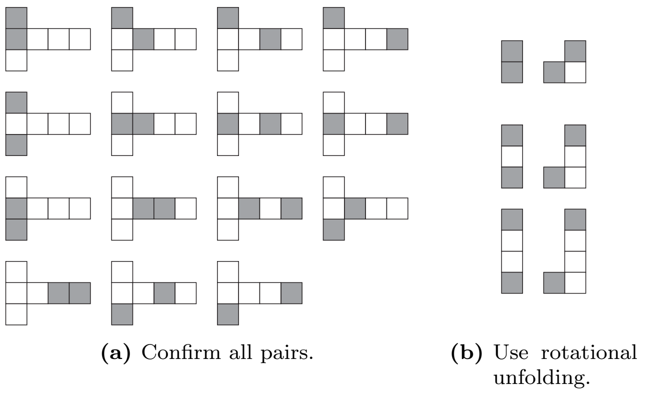


Figure 7: The thick lines form a tree that includes all the lattice cube's vertices (the starred ones).



(a) Confirm all pairs. **(b)** Use rotational unfolding.

Figure 8: Pairs of squares must be checked for overlap in a certain edge unfolding.

2.5 Methods for checking the overlap

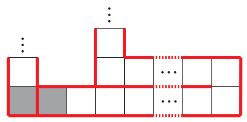
Herein, we introduce a method for verifying the non-existence of overlapping edge unfoldings used in the previous research [14].

To prove that there is no edge unfolding with overlap for a given polyhedron, we must check the overlapping for all pairs of faces of all edge unfoldings. For example, a cube has 11 edge unfoldings, and each unfolding has ${}_6C_2 = 15$ pairs of squares that need to be checked for overlap (see Figure 8(a)). However, focusing on the symmetry of relative positions, the number of pairs that actually need to be checked is six, as shown in Figure 8(b). In other words, if we confirm that none of them overlap, we can conclude that all edge unfoldings do not overlap.

An algorithm called rotational unfolding has been developed with a focus on this point [14]. The method of rotational unfolding enumerates minimum unfoldings containing two faces for each pair of faces of the polyhedron. In the method, each unfolding is drawn by "Rolling the polyhedron on a plane from the state that one face is bottom to the state that another is bottom." This method greatly reduces the number of checking patterns for overlap. For details on rotational unfolding, refer to [14].

3 Results

This study presents the following theorem for cuboids:


(a) An unnecessary path in rotation unfolding.

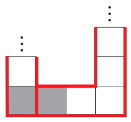

(b) A desired path.

Figure 9: The pair of gray faces of the path (a) do not need to be considered because they have already checked in the path (b).

Theorem 8

- *For the* $(1,1,1)$*-cuboid or* $(\sqrt{2},\sqrt{2},\sqrt{2})$*-cuboid, there is no overlapping lattice unfolding.*
- *For the* $(1,1,2)$*-cuboid, there are no faces-in-touch lattice unfoldings and no edges-in-touch lattice unfoldings, but a vertices-in-touch lattice unfolding exists.*
- *For the* $(1,2,2)$*-cuboid or* $(2,2,2)$*-cuboid, there is no faces-in-touch lattice unfoldings, but edges-in-touch lattice unfolding and vertices-in-touch lattice unfolding exist.*
- *For any other cases, faces-in-touch lattice unfoldings, edges-in-touch lattice unfoldings, and vertices-in-touch lattice unfoldings exist.*

First, we show the method to check the overlapping of lattice unfoldings by computational experiment. By implementing the following method, we check the non-existence side of the statements of Theorem 8. This experiment includes the verification of the previous results [7] and [15].

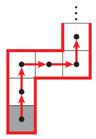
3.1 The method to check the overlapping of lattice unfoldings by computational experiment

The method of rotational unfolding in Section 2.5 is used to enumerate edge unfoldings, but cannot be directly used for lattice unfolding. This section shows the method of extending rotational unfolding to lattice unfolding.

In the rotational unfolding, the dual graph $D(G_P)$ of its edge representation graph G_P is used for technical reasons. According to this, we consider the dual graph $D(G_C)$ of the lattice representation graph G_C for the lattice unfolding of a cuboid C. Lemma 7 implies that G_C has no leaf nodes other than the vertices of the cuboid. When enumerating $D(G_C)$, it is necessary to remove redundant parts of the path, such as the one shown in Figure 9(a). Therefore, we introduce the following characters for information about the "direction of rolling when viewed from one step before":

R : Roll to the right from one step before.

C : Roll straight from one step before.


(a) "CCRCL"

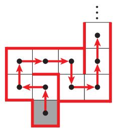
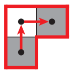
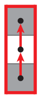
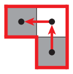
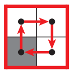

(b) "CLRRCRLLC"

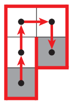
Figure 10: Strings of paths obtained by rotational unfolding.


(a) "CR"


(b) "CC"


(c) "CL"


(d) "CRRR"


(e) "CCRR"

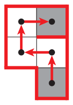

(f) "CLRR"

Figure 11: (a)-(c): Paths for steps 2. (d)-(f): Paths for steps 3 and 4 when rolling "RR".

L : Roll to the left from one step before.

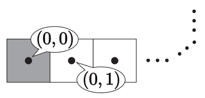
In addition, the path obtained using the rotational unfolding will be represented as a string (Figure 10 shows examples). In the rotational unfolding, the first step is to roll straight ahead, so the path obtained in the first step is "C". We can state the following lemma:

Lemma 9 *When representing the path obtained by rotational unfolding as a string, it includes redundant parts if it contains "RR" or "LL".*

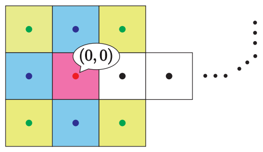
Proof. In the second step of the rotational unfolding, we have three cases: (1) rolling to the right (string: "CR", Figure 11(a)), (2) rolling straight (string: "CC", Figure 11(b)), and (3) rolling to the left (string: "CL", Figure 11(c)). If we repeat the action of rolling right, or "RR", twice after the second step, we get (1) "CRRR" (Figure 11(d)), (2) "CCRR" (Figure 11(e)), and (3) "CLRR" (Figure 11(f)). For case (1), this situation cannot occur because we have already used the face as part of the constructed path. For cases (2) and (3), the paths represented by "CR" and "CC" (Figures 11(a) and 11(b)) have already been checked for overlap. Therefore, if "RR" is included in the path, it contains redundant parts. Similarly, we can show this in the case of "LL". □

When a cuboid has an overlapping lattice unfolding, we can determine how they overlap using the following observation:

Observation 10 *In rotational unfolding, compute the center coordinates of the face at one endpoint, assuming its center coordinates are* $(0,0)$*. Then, while rolling the*



(a) The coordinates of the center of each face.



(b) The coordinates of the center of the face at the other endpoint.

Figure 12: The method to check for overlap in rotational unfoldings, including their type.

path sequentially, compute the center coordinates of the face at the other endpoint (see Figure 12(a)). We can determine the type of unfolding based on the coordinates of the center of the face at the other endpoint:

- *If the coordinate is* $(0, 0)$*, it is a faces-in-touch unfolding (a red face* 🟥 *in Figure 12(b)).*
- *If the coordinates are* $(0, 1)$*,* $(-1, 0)$*, or* $(0, -1)$*, it is an edges-in-touching unfolding (blue faces* 🟦 *in Figure 12(b)).*
- *If the coordinates are* $(1, 1)$*,* $(1, -1)$*,* $(-1, -1)$*, or* $(-1, 1)$*, it is a vertices-in-touch unfolding (green faces* 🟨 *in Figure 12(b)).*

3.2 Construction of specific overlapping unfoldings

Hereafter, we prove the existence side of the statements of Theorem 8 by showing specific overlapping unfoldings.

3.2.1 Case of $L = 1$

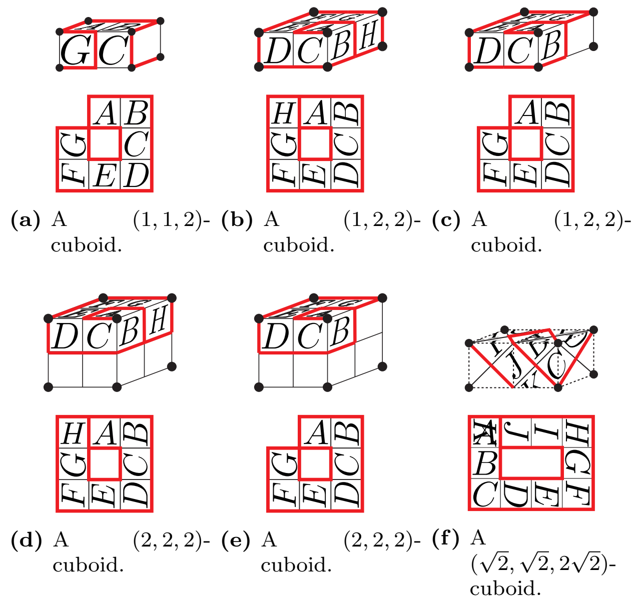
From Theorem 2 and Theorem 3, a faces-in-touch, an edges-in-touch, and vertices-in-touch unfoldings exist for the (x, y, z)-cuboid, where $z \geq 3$. For the remaining cases for the case of $L = 1$, we provide specific examples of unfoldings as follows:

Lemma 11

- *A vertices-in-touch unfolding exists for the* $(1, 1, 2)$*-cuboid, as shown in Figure 13(a).*
- *An edges-in-touch unfolding and a vertices-in-touch unfolding exist for the* $(1, 2, 2)$*-cuboid, as shown in Figure 13(b), and Figure 13(c) respectively.*
- *An edges-in-touch unfolding and a vertices-in-touch unfolding exist for the* $(2, 2, 2)$*-cuboid, as shown in Figure 13(d), and Figure 13(e) respectively.*

3.2.2 Case of $L = \sqrt{2}$, $L = \sqrt{5}$, and $L = \sqrt{10}$

From the inclusion relationship between the edges-in-touch and vertices-in-touch unfolding, we have only to show the existence of the faces-in-touch unfolding.



(a) A $(1, 1, 2)$-cuboid. (b) A $(1, 2, 2)$-cuboid. (c) A $(1, 2, 2)$-cuboid. (d) A $(2, 2, 2)$-cuboid. (e) A $(2, 2, 2)$-cuboid. (f) A $(\sqrt{2}, \sqrt{2}, 2\sqrt{2})$-cuboid.

Figure 13: Overlapping lattice unfolding by cutting along the red lines.

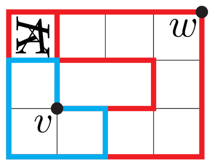


Figure 14: The lattice unfolding Q_L.

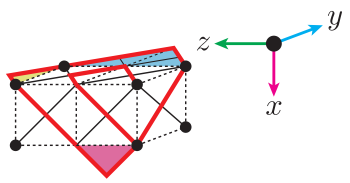


Figure 15: The lattice unfolding Q_L can be embedded in the three faces in front of the $(\sqrt{2}, \sqrt{2}, 2\sqrt{2})$-cuboid.

A faces-in-touch unfolding exist for the $(\sqrt{2}, \sqrt{2}, 2\sqrt{2})$-cuboid, as shown in the lower part of Figure 13(f). From now on, we will refer to this lattice unfolding as Q_L (Figure 14). Moreover, the $(\sqrt{2}, \sqrt{2}, 2\sqrt{2})$-cuboid can be unfolded to partially include the lattice unfolding Q_L, because Q_L can be embedded on the three faces in front of the $(\sqrt{2}, \sqrt{2}, 2\sqrt{2})$-cuboid. Note that we have to fold the three triangular faces indicated in pink (🟥), light blue (🟦), and light green colors (🟨) in the positive direction of the y-axis, the positive direction of the x-axis, and the positive direction of the x-axis, respectively (see Figure 15). This embedding method can also be applied to the $(x\sqrt{2}, y\sqrt{2}, z\sqrt{2})$-cuboid, where $x, y, z \geq 2$, as shown in Figure 16.

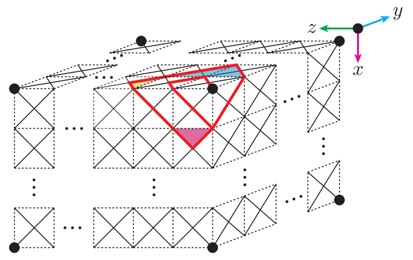
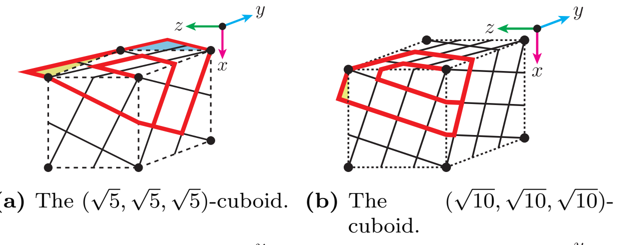
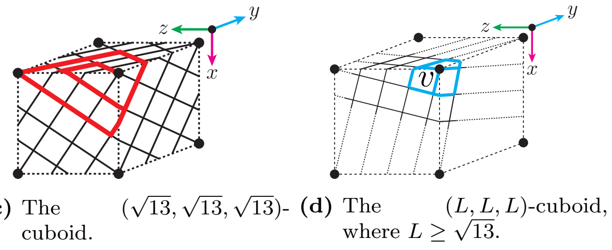


Figure 16: The lattice unfolding Q_L can be embedded in the $(x\sqrt{2}, y\sqrt{2}, z\sqrt{2})$-cuboid, where $z \leq 2$-cuboid.



(a) The $(\sqrt{5}, \sqrt{5}, \sqrt{5})$-cuboid. **(b)** The $(\sqrt{10}, \sqrt{10}, \sqrt{10})$-cuboid.



(c) The $(\sqrt{13}, \sqrt{13}, \sqrt{13})$-cuboid. **(d)** The (L, L, L)-cuboid, where $L \geq \sqrt{13}$.

Figure 17: The lattice unfolding Q_L can be embedded in each cuboid.

The same embedding can be performed for cases where $L = \sqrt{5}$ and $L = \sqrt{10}$ (refer to Figure 17).

3.2.3 Case of $L \geq \sqrt{13}$

The lattice unfolding Q_L can be embedded in the $(\sqrt{13}, \sqrt{13}, \sqrt{13})$-cuboid, as shown in Figure 17(c). Here, we present the following lemma:

Lemma 12 *The lattice unfolding Q_L can be embedded in the (L, L, L)-cuboid, where $L \geq \sqrt{13}$.*

Proof. Consider three unit squares with vertex v in common (Figure 17(d)). The three-unit squares enclosed in blue in Figure 14 can be embedded in this point. Let S be the side face of a cone with the length of the axis $\sqrt{13}$ and central angle 270° (Figure 18). Hereafter S is called the *cone*. Since the central angle of the cone S is 270°, the three unit squares enclosed in blue in Figure 14 can be embedded with vertex v coinciding. Additionally, due to the Euclidean distance between the point v and the furthest point w in Figure 14 being $\sqrt{2^2 + 3^2} = \sqrt{13}$, the remaining faces except for the three faces enclosed in blue can be embedded as shown in Figure 18 (right). The cone S can be embedded in the three front faces of a (L, L, L)-cuboid where

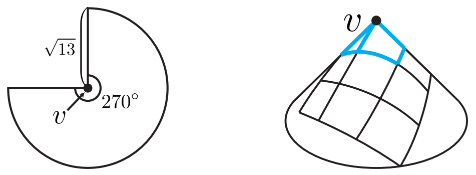


Figure 18: The side face of a cone with the length of the axis $\sqrt{13}$ and central angle 270°. By rounding the left fan shape, the right solid is obtained. We can embed a lattice unfolding Q_L in this.

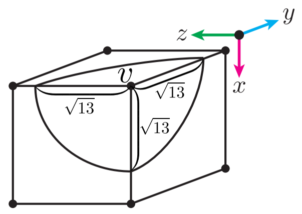


Figure 19: A cone S can be embedded in the three faces in front of the (L, L, L)-cuboid, where $L \geq \sqrt{13}$.

$L \geq \sqrt{13}$, as shown in Figure 19. From the fact that the cone S can be embedded on a (L, L, L)-cuboid and that the lattice unfolding Q_L can be embedded on top of the cone S, it can be concluded that the lattice unfolding Q_L can be embedded on the three front faces of a (L, L, L)-cuboid. □

From this lemma, a faces-in-touch unfolding exists for the (xL, yL, zL)-cuboid in any of the x, y, z, where $L \geq \sqrt{13}$. The same can be said for edges-in-touch and vertices-in-touch unfolding due to the inclusion relationship.

4 Conclusion

In this paper, we completely clarified the condition for an unfolding to have an overlapping when we unfold a cuboid into a polyomino. This result gives us a boundary condition for whether the unfolding of a polyhedron has overlap, depending on the fineness of the cut lines. This result could be immediately extended to the triangular lattice unfolding of an octahedron or icosahedron.

Moreover, our technique in Section 3.1 would be useful for more general ways of unfolding; for example, the case to allow cutting diagonals of the faces of convex regular-faced polyhedra. This approach would also be important, as it would provide more information than Table 1 about the conditions that an unfolding has overlaps.

Acknowledgments.

This work was supported in part by JSPS KAKENHI Grant JP20H05961, 20H05964 , 22H01423, and JP22KJ1480.

References

[1] B. Aronov and J. O'Rourke. Nonoverlap of the star unfolding. *Discrete Comput. Geom.*, 8:219–250, 1992.

[2] T. C. Biedl, E. D. Demaine, M. L. Demaine, A. Lubiw, M. H. Overmars, J. O'Rourke, S. Robbins, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. In *10th Canadian Conference on Computational Geometry*, 1998.

[3] H. T. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry*. Springer-Verlag, reissue edition, 1991.

[4] A. Dürer. Underweysung der messung, mit dem zirckel und richtscheyt in linien ebenen unnd gantzen corporen, 1525. Available electronically for example http://books.google.com/books?id=5fxOAAAAcAAJ.

[5] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007.

[6] B. Grünbaum. Are your polyhedra the same as my polyhedra? In *Discrete and Computational Geometry*, volume 25, pages 461–488. Springer, 2003.

[7] R. Hearn. personal communication, 2018.

[8] K. Hirose. Hanseitamentai no tenkaizu no kasanari ni tsuite (On the overlap of Archimedean solids), in Japanese, 2015. Saitama Univ. graduation thesis. Supervisor : Takashi Horiyama.

[9] T. Horiyama and W. Shoji. Edge unfoldings of platonic solids never overlap. In *23rd Canadian Conference on Computational Geometry*, 2011.

[10] J. Mitani and R. Uehara. Polygons folding to plural incongruent orthogonal boxes. In *20th Canadian Conference on Computational Geometry*, 2008.

[11] M. Namiki and K. Fukuda. Unfolding 3-dimensional convex polytopes. *A package for Mathematica 1.2 or 2.0. Mathematica Notebook*, 1993.

[12] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15(1):193–215, 1986.

[13] T. Shiota. Overlapping edge unfoldings for convex regular-faced polyhedrons, 2023. Kyushu Institute of Technology master's thesis. Supervisor : Toshiki Saitoh.

[14] T. Shiota and T. Saitoh. Overlapping edge unfoldings for archimedean solids and (anti)prisms. In *17th International Conference and Workshops of Algorithms and Computation*, volume 13973 of *LNCS*, pages 36–48. Springer, 2023.

[15] H. Sugiura. personal communication, 2018.

[16] T. Uno. personal communication, 2008.

A Parameterized Algorithm for Flat Folding

David Eppstein*

Abstract

We prove that testing the flat foldability of an origami crease pattern (either labeled with mountain and valley folds, or unlabeled) is fixed-parameter tractable when parameterized by the ply of the flat-folded state and by the treewidth of an associated planar graph, the cell adjacency graph of an arrangement of polygons formed by the flat-folded state. For flat foldings of bounded ply, our algorithm is single-exponential in the treewidth; this dependence on treewidth is necessary under the exponential time hypothesis.

1 Introduction

In a foundational result in the computational complexity of mathematical paper folding, Bern and Hayes proved in 1996 that it is NP-complete to determine whether a crease pattern, described as a set of straight fold lines on a flat piece of paper, can be folded to lie flat again after exactly the prescribed folds have been made [5]. This result holds regardless of whether the folds are given purely as line segments, or whether they additionally specify whether each fold is to be a mountain fold or a valley fold. It assumes a general model of folding where only the existence of the desired folded state is to be determined, and not a sequence of motions that reach it, but subsequent work has also proved similar hardness results for other models such as *box pleating*, where the folds are aligned with the axes and diagonals of a square grid [2], and the *simple folding* typical of sheet-metal manufacturing in which this motion must only be made on one fold line at a time [3, 4].

On the positive side, not much is known about classes of crease patterns for which foldability is easier to determine. One such class, but a very limited one, is the class of patterns where the folds meet in a single vertex (or as a degenerate case, where they all lie on parallel lines). In this case, a linear-time greedy algorithm follows from the big-little-big lemma, in which creases forming a sharp angle between two wider angles must fold in a fixed way, allowing a reduction to a simpler configuration [5]. Two more polynomial cases are simple folding of rectangles subdivided into congruent rectangles ("map folding") [4], and general map folding of $2 \times n$ grids of rectangles [21].

In this work, we provide the first algorithmic upper bounds on testing flat foldability of arbitrary crease patterns, not restricted to special cases such as map folding. Our work analyzes this problem using tools from parameterized complexity. We show that flat-foldability is *fixed-parameter tractable* when parameterized by two values: the *ply* of the crease pattern (how many layers of paper can overlap at any point of the flat-folded result), and the *treewidth* of an associated *cell adjacency graph* constructed by overlaying the flat polygons of the crease pattern in the positions they would take in their folded state. The pattern may either be labeled with mountain and valley folds or unlabeled. We identify a wide class of patterns for which flat foldability is easy: those with bounded ply and bounded treewidth. For flat foldings of bounded ply, our algorithm is single-exponential in the treewidth. As we show in an appendix, this exponential dependence is necessary under the exponential time hypothesis, both for unlabeled and labeled crease patterns. We do not have as strong an argument for why the dependence on ply is necessary, but if it could be eliminated, we could solve map folding in polynomial time, a major open problem in this area.

Bounded ply is natural in paper folding, as large ply can lead to difficulty in the physical realization of a folding [12]. The treewidth parameter is intended to capture the notion of a crease pattern that is complicated only in one dimension, and simple in a perpendicular dimension, as occurs (with large ply) for $2 \times n$ map folding. Single-vertex crease patterns also automatically have low treewidth (their cell adjacency graph is just a cycle; see Section 2.3) but may again have high ply. Fixed-parameter tractability of an algorithm means that its worst-case time bound has the form of a polynomial in the input size, multiplied by a non-polynomial function of the parameters; in our case this function is factorial in the ply and exponential in the treewidth. On inputs for which the parameters are bounded, this function value is also bounded and the time bound simplifies to being purely a polynomial of the input size.

Another class of example patterns for which the parameters of our algorithm are naturally bounded comes from the origami font of Demaine, Demaine, and Ku [8–10]. Rendering text in this font converts it into an origami crease pattern (Fig. 1). When folded, this pattern produces a three-dimensional structure consisting of letterform-shaped vertical walls on a flat background surface (Fig. 2). The resulting structures are not ac-

*Department of Computer Science, University of California, Irvine. Research supported in part by NSF grant CCF-2212129.

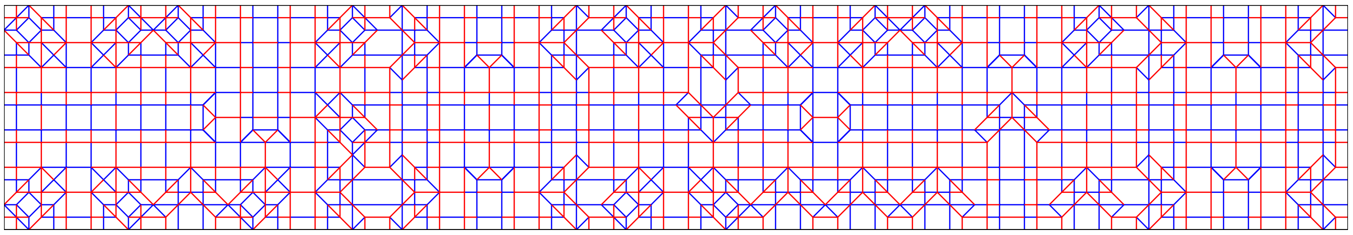


Figure 1: A crease pattern for the origami font of Demaine, Demaine, and Ku, produced by `http://erikdemaine.org/fonts/maze/?text=origami`.

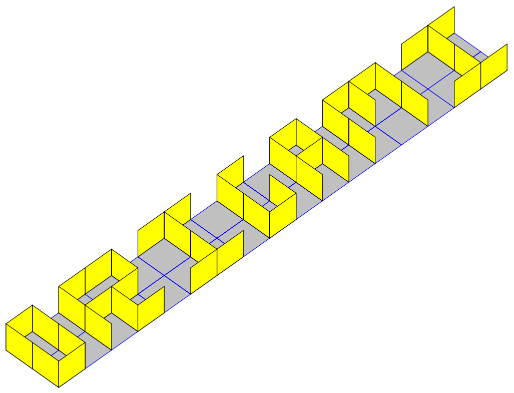


Figure 2: The 3d folded form of the pattern from Fig. 1, as produced by `http://erikdemaine.org/fonts/maze/?text=origami`.

tually flat foldings (because of the vertical walls) but can easily be modified to be. The resulting crease pattern, for a line of text, has bounded ply, high complexity along any horizontal line through the pattern, and low complexity along any vertical line. Its cell adjacency graph has bounded bandwidth, but for a modified version of the font that included ascenders and descenders it would instead have bounded pathwidth, both of which are special cases of our bounded treewidth assumption.

2 Preliminaries

2.1 Flat folding

Following our previous work [15], we base our definition of flat folding on a *local flat folding*, a simplified model of folding which describes only how the folding maps a flat surface to itself, and does not describe the spatial arrangement of the layers of paper as a flat-folded surface. We will then augment this model to include layer ordering, to define a *flat folding*.

Thus, we define a *local flat folding* of a planar polygon P to be a *continuous piecewise isometry* φ from P to the plane. That is, it is a continuous function that acts as a distance-preserving mapping of the plane within each of a system of finitely many interior-disjoint polygons whose union is P. The points at which φ is not locally an isometry lie on the boundaries of these polygons, forming *creases* (line segments between two polygons mapped differently by φ) and *vertices* (points where multiple creases meet). We may choose the polygons of φ so that each polygon is bounded by creases and by the boundary of P. The *crease pattern* of a local flat folding is this system of creases and vertices. At this level of detail, there is no distinction between mountain folds and valley folds.

Observation 1. *Given a decomposition of a polygon P into smaller polygons, we can determine in linear time whether this decomposition forms the crease pattern of a local flat folding, and if so reconstruct a function φ having that decomposition as its crease pattern.*

Proof. We choose an arbitrary starting polygon, set φ to be the identity within this polygon, and then traverse the adjacencies between polygons of the decomposition. When we traverse the edge between a polygon whose mapping under φ has been determined to another polygon whose mapping has not, we set the mapping for the new polygon to be the mapping for the old polygon, reflected across the line through the traversed edge. When we traverse an edge to a polygon whose mapping has already been determined, we check that its mapping is consistent with this reflection. □

The function φ, constructed in this way, is unique up to rigid transformations of the plane.

We define the *arrangement* of a local flat folding to be the result of overlaying the transformed copies of each of its polygons. It partitions the plane into *cells*, polygons that are not crossed by the image of any crease. Within each cell, all points have preimages coming from the same set of polygons of the crease pattern. The *ply* of a cell is the number of these preimages, and the ply of the crease pattern is the maximum ply of any cell. See Fig. 3. Using standard methods from computational geometry, an arrangement of a local flat folding with n creases has complexity $O(n^2)$ and can be constructed (including the calculation of its ply) in time $O(n^2)$.

Our previous work [15] defined a *global flat folding* to be "a local flat folding that, for every $\varepsilon > 0$, is ε-

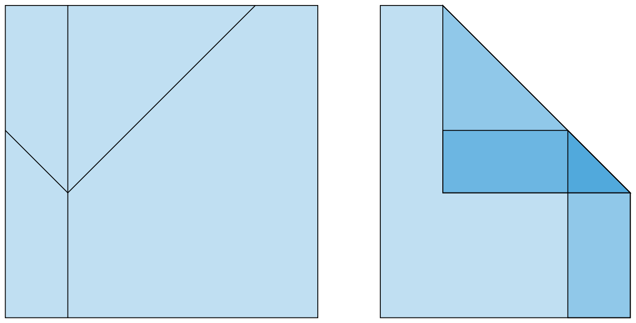


Figure 3: The crease pattern of a local flat folding (left) and the arrangement of the folding (right), with shading indicating the ply of each arrangement cell. The ply of the overall pattern is four, equal to the maximum ply in the small triangular cell.

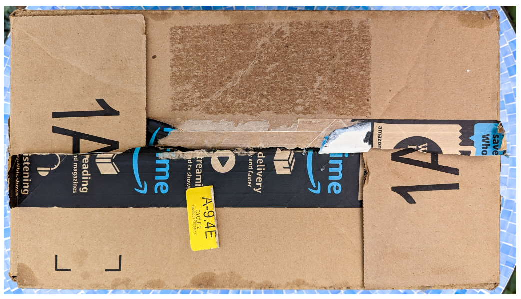


Figure 4: Cyclically-ordered box-top flaps

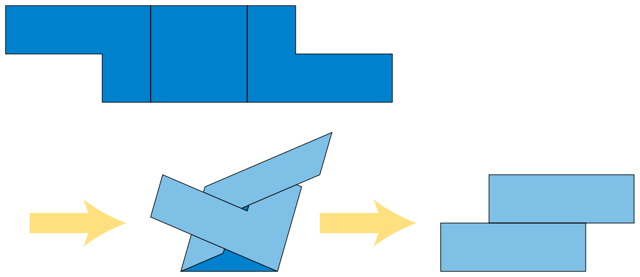


Figure 5: A crease pattern with two valley folds that, when flat-folded, causes its two L-shaped polygons to have two different above-below orderings in the two cells of the arrangement where they overlap.

close to a topological embedding of the plane into three-dimensional space", but for our purposes we need to actually describe the three-dimensional embedding combinatorially, not merely to assert its existence. Instead, we define a *layering* of a local flat folding to be an assignment, for each cell of the arrangement of the folding, of a vertical ordering on the preimage polygons of the cell. We allow different cells to have different and inconsistent vertical orderings. This may be necessary to model real-world foldings in which the vertical ordering of polygon has cycles, as happens for instance in flexagons [17] and in a common method for folding the four flaps of a box top (Fig. 4). It is even possible for the same two polygons of a crease pattern to have two different above-below orderings in two different cells of the arrangement in which they overlap (Fig. 5).

We define a *flat folding* to be a local flat folding together with a layering that, for every $\varepsilon > 0$, is consistent with the layering coming from a topological embedding of the crease pattern into three-dimensional space that is ε-close to the local flat folding. Here, "close" means there exists a local flat folding into a plane in space so that, for every point of the crease pattern, its images under the topological embedding and under the local flat folding have distance at most ε from each other. To avoid topological difficulties we additionally require that a line perpendicular to the plane, through a point of the plane farther than ε from any crease, has exactly one point of intersection with each polygon in the topological embedding: the embedding cannot be "crumpled" far from its creases. With this restriction, the polygons that map to each cell have a consistent layering, the ordering in which they meet any such perpendicular line.

If we look at a cross-section of such a topological embedding, across any crease of the embedding, we will see the layers in two adjacent cells of the arrangement. Two layers in the same cell can be paired up to form a crease, two layers from the two cells can be paired up to form parts of a polygon that span the cell without forming a crease, and it is also possible to have an unpaired layer whose boundary at the crease coincides with a boundary of the overall crease pattern (Fig. 6, left). These layers and pairs of layers must meet certain obvious conditions:

- If two polygons span the two cells without being creased, they must be consistently ordered in both cells instead of crossing at the crease (Fig. 6, top right).
- If two layers of the same cell meet in a crease, and another polygon spans the two cells without being creased, the polygon cannot lie between the two creased layers of the first polygon, as their crease would block it from extending into the second cell (Fig. 6, middle right).
- If two pairs of layers in the same cell meet in the same crease, then their layers cannot alternate, as this would again form a crossing (Fig. 6, bottom right). However, it may be possible to have alternating pairs of layers that meet in different creases, along different edges of the same cell.
- If two layers of the same cell meet in a crease, and are labeled as being a mountain fold or valley fold in the crease pattern, then the ordering of the layers must be consistent with that type of fold (not shown).

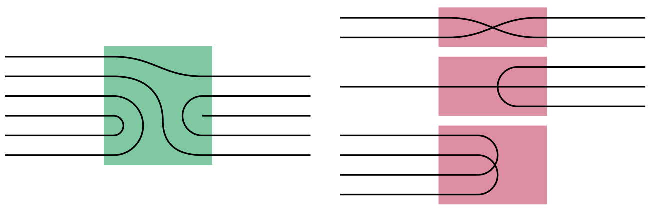


Figure 6: Left: cross-section through a crease (shaded region) of a uncrossed layering. Right: Three ways that a layering can be inconsistent across a crease: two uncreased polygons cross (top), an uncreased polygon is blocked by two layers that connect to form a crease (middle), or two pairs of creased layers cross (bottom).

We define a layering for a local flat folding to be *uncrossed* when, at each crease, it meets all of these conditions.

Lemma 2. *A local flat folding comes from a flat folding if and only if it has an uncrossed layering.*

Proof. In one direction, if a flat folding exists, it cannot violate any of the conditions above, because each describes a certain type of crossing, and topological embeddings forbid crossings. In the other direction, every uncrossed layering comes from a flat folding: one can form a 3d embedding from it, by shrinking each cell a small distance from its boundary, making parallel copies of the cell in 3d in the order given by the layering, all separated from each other but within distance ε of the plane of the local flat folding, and connecting them with curved patches of surface near each crease.

It is unnecessary to add more case analysis for the way layerings can interact at a vertex, instead of across a crease. Two surfaces in 3d cannot cross each other at a single point, without crossing along a curve touching that point, so if a system of surfaces in 3d defined from a uncrossed layering avoids crossings except at points ε-close to the vertices, it can be converted into a topological embedding for the same layering that avoids crossing everywhere. □

2.2 Treewidth

A *tree decomposition* of a graph G consists of an unrooted tree T, and an assignment to each tree vertex t_i of a set B_i of vertices from G (called a *bag*), such that each vertex of G belongs to the bags from a connected subtree of T, and each edge of G has endpoints that belong together in at least one bag. Its *width* is the maximum size of a bag, minus one, and the *treewidth* of G is the minimum width of any tree decomposition of G. Many optimization problems that are hard on arbitrary graphs can be solved in linear time on graphs of bounded treewidth, using dynamic programming over their tree decompositions. Although finding the treewidth is itself a hard optimization problem, it can be solved in linear

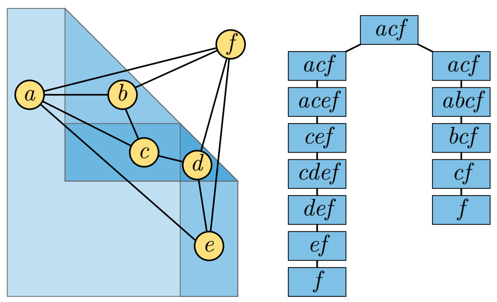


Figure 7: The arrangement from Fig. 3, its cell adjacency graph, and a nice tree decomposition of the cell adjacency graph.

time for graphs of bounded treewidth, with a time bound that is exponential in the cube of the width [6]. In our application we will be using the treewidth of planar graphs, derived from the arrangement of a crease pattern. It is unknown whether planar treewidth is hard, but it can be approximated in (unparameterized) polynomial time with an approximation ratio of 3/2 by an algorithm for a closely related width parameter called *branchwidth* [23].

It will simplify the description of our algorithm to use a tree decomposition of a special form, called a *nice tree decomposition*. This differs from a tree decomposition in being a rooted tree. The tree vertices and their bags have four types:

- *Leaf bags*, leaves of the rooted tree, have exactly one graph vertex in the bag.
- *Introduce bags* have exactly one child vertex in the tree, and their bag differs from that of the child by the addition of exactly one graph vertex.
- *Forget bags* have exactly one child vertex in the tree, and their bag differs from that of the child by the removal of exactly one graph vertex.
- *Join bags* have exactly two children, whose bags are both equal to the join bag.

A nice tree decomposition can be constructed in linear time from an arbitrary tree decomposition, without increasing the width, and it has size linear in the size of the input tree decomposition [20].

2.3 Cell adjacency graphs and their treewidth

Recall that our definition of flat folding involves constructing an arrangement of polygons, the images of the polygons in the crease pattern under the mapping that defines a local flat folding. The usual notion of an *arrangement graph* is a planar graph with a vertex for each crossing or endpoint of a line segment in this arrangement, and an edge for each piece of polygon boundary connecting two of these vertices [7]. Instead, we use its dual graph, which we call the *cell adjacency graph.* This has a vertex for each cell of the arrangement, and an edge between each two neighboring cells. It has been used before in computational geometry (e.g. [14]), but appears to lack a standard name.

Even when a cell has ply zero, we include it in this graph, in order to check for crossings along the creases between this cell and its neighbors. For example, in the map folding problem, a square grid crease pattern is folded down to a single square, but the arrangement has two cells, the inside of the square and the outside, so the cell adjacency graph is K_2. In the case of a single-vertex crease pattern, the local flat folding produces an arrangement consisting of wedges all having this vertex as their apex, and its cell adjacency graph is a cycle.

The two main parameters for the analysis of our algorithm will be the ply of the local flat folding, and the treewidth of the cell adjacency graph. Fig. 7 depicts an example of a cell adjacency graph of treewidth 2, and a nice tree decomposition with a join bag at its root.

3 The algorithm

We will test the flat foldability of a crease pattern by first attempting to construct its local flat folding. If this step fails, a flat folding does not exist, and our algorithm exits with a negative answer. Next, we construct its arrangement and its cell adjacency graph, find an optimal or near-optimal tree decomposition of the cell adjacency graph using any of the various algorithms known for this problem, and convert the tree decomposition to a nice tree decomposition of the same width.

Finally, we reach the main part of our algorithm: a bottom-up dynamic program on the bags of the tree decomposition. If B is any bag (that is, a set of cells of the arrangement, associated with a vertex of the nice tree decomposition), we define a *state* of B to be a layering of each cell in B.

Observation 3. *In a tree decomposition of width w for a crease pattern of ply p, every bag has at most $(p!)^{w+1}$ states.*

If B has a child C in the tree decomposition, then we say that a state of B is *consistent* with a state of C if they have the same layering in all of the cells that belong to both bags. We say that a state of bag B is *locally uncrossed* if, for all pairs of adjacent cells that both belong to B, the layerings of these two cells in this state meet the same conditions that we used earlier to define a global layering as being uncrossed. We say that a state is *valid* when it is locally uncrossed and is consistent with (recursively defined) valid states for all child bags.

Lemma 4. *For any bag B of the tree decomposition, there exists a valid state for B if and only if there exists a layering for the entire local flat folding that meets the conditions of being uncrossed at all creases between pairs of cells that occur together in B or one of its descendants in the tree decomposition.*

Proof. If such a layering exists, its restriction to the cells in B and its descendant bags produces a valid state. If a valid state exists, coming from a recursively constructed set of valid states among its descendant bags, then each of these states must consistently layer the cells that they have in common, by the requirement of tree-decompositions that each graph vertex belong to bags in a connected subtree. Form a global layering by choosing arbitrarily a layering for each cell that is not included among these descendants. Then it must be uncrossed at all creases between pairs of cells that occur together in B or one of its descendants, because any crossing would cause the state to be invalid at that bag, violating the assumption that we have a recursively constructed set of valid states. $\square$

Lemma 5. *If we have already computed the valid states of each child of a given bag B of a nice tree decomposition, we can compute the valid states for B itself in time $O(pw(p!)^{w+1})$.*

Proof. We apply a case analysis according to the type of B in the decomposition.

- At a leaf bag, all states are valid, because there are no creases between pairs of cells to cause crossings.
- At an introduce bag, we must add a layering for the introduced cell to all valid layerings of the other cells from the child node. For each child layering, and each layering of the introduced cell, we check at most w previously-unrepresented creases, each in time $O(p)$, to determine whether it forms any of the forbidden crossing types.
- At a forget bag, all valid states of the child node determine a valid state of the bag, by forgetting the layering on the cell that is not included.
- At a join bag, a state is valid when it is valid in both child states. We can intersect the sets of valid states in both children, in time linear in the number of possible states, using a bit array. $\square$

Putting these pieces together gives our main result:

Theorem 6. *Testing flat foldability of a crease pattern with n creases and ply p, with a cell adjacency graph of treewidth w, can be performed in time that is fixed-parameter tractable in p and w, and quadratic in n.*

Proof. We construct the nice tree decomposition as described above, and traverse it in bottom-to-top order, using Lemma 5 to determine the valid states in each bag. A folding exists if and only if there is a valid state at the root bag, by Lemma 4. The quadratic dependence on n comes from the size of the arrangement of the local flat folding, and the size of the tree decomposition of its cell adjacency graph. The dependence on ply and width comes from the time bound per bag in Lemma 5, the time to construct a tree decomposition using known algorithms, and the relation between the width of the cell adjacency graph and the width of the constructed decomposition coming from the choice of these algorithms. □

4 Conclusions

We have shown that flat foldability, in a general model allowing cyclic overlaps between polygons, can be tested in fixed-parameter tractable time when parameterized both by ply and by the treewidth of an associated cell adjacency graph. Both parameters appear necessary for this result: the known NP-hardness reductions for flat foldability can be made to have bounded ply (but unbounded treewidth), while the still-open map folding problem has bounded treewidth and more strongly bounded cell adjacency graph size (but unbounded ply).

It would be of interest to extend our algorithms to other forms of flat folding, such as the *simple folding* models [3, 4]. Another direction for possible future work concerns models of folding that require the existence of a three-dimensional continuous motion respecting the given fold lines (rigid origami [1, 22, 24]), as well as inputs where the desired folded state is in some way three-dimensional (such as the raised ridges in the origami fonts of Demaine, Demaine, and Ku [8–10]. Although there has been extensive study of types of instance that can or cannot be guaranteed to have a continuous motion taking them between their unfolded and folded states [11, 13, 16], there is little work on algorithmic time bounds for testing the existence of this sort of motion. Whether these three-dimensional models of origami can be reduced to a combinatorial problem to which the sort of methods described here can apply remains a challenge.

References

[1] Zachary Abel, Jason Cantarella, Erik D. Demaine, David Eppstein, Thomas Hull, Jason S. Ku, Robert J. Lang, and Tomohiro Tachi. Rigid origami vertices: Conditions and forcing sets. *J. Comput. Geom.*, 7(1):171–184, 2016. doi:10.20382/jocg.v7i1a9.

[2] Hugo A. Akitaya, Kenneth C. Cheung, Erik D. Demaine, Takashi Horiyama, Thomas Hull, Jason S. Ku, Tomohiro Tachi, and Ryuhei Uehara. Box pleating is hard. In Jin Akiyama, Hiro Ito, Toshinori Sakai, and Yushi Uno, editors, *Discrete and Computational Geometry and Graphs – 18th Japan Conference, JCDCGG 2015, Kyoto, Japan, September 14–16, 2015, Revised Selected Papers*, volume 9943 of *Lecture Notes in Comput. Sci.*, pages 167–179. Springer, 2015. doi:10.1007/978-3-319-48532-4_15.

[3] Hugo A. Akitaya, Erik D. Demaine, and Jason S. Ku. Simple folding is really hard. *J. Information Processing*, 25:580–589, 2017. doi:10.2197/ipsjjip.25.580.

[4] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S. B. Mitchell, Saurabh Sethia, and Steven S. Skiena. When can you fold a map? *Comput. Geom. Theory & Appl.*, 29(1):23–46, 2004. doi:10.1016/j.comgeo.2004.03.012.

[5] Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms (SODA '96)*, pages 175–183, Philadelphia, PA, 1996. Society for Industrial and Applied Mathematics. URL: https://portal.acm.org/citation.cfm?id=313852.313918.

[6] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

[7] Prosenjit Bose, Hazel Everett, and Stephen Wismath. Properties of arrangement graphs. *Internat. J. Comput. Geom. Appl.*, 13(6):447–462, 2003. doi:10.1142/S0218195903001281.

[8] Erik D. Demaine and Martin L. Demaine. Fun with fonts: algorithmic typography. *Theor. Comput. Sci.*, 586:111–119, 2015. doi:10.1016/j.tcs.2015.01.054.

[9] Erik D. Demaine and Martin L. Demaine. Adventures in maze folding art. *J. Information Processing*, 28:745–749, 2020. doi:10.2197/ipsjjip.28.745.

[10] Erik D. Demaine, Martin L. Demaine, and Jason S. Ku. Origami maze puzzle font. In *Exchange Book of the 9th Gathering for Gardner (G4G9)*. March 24–28 2010. URL: https://erikdemaine.org/papers/MazeAlphabet_G4G9/.

[11] Erik D. Demaine, Satyan L. Devadoss, Joseph S. B. Mitchell, and Joseph O'Rourke. Continuous foldability of polygonal paper. In *Proceedings of the 16th Canadian Conference on Computational Geometry, CCCG'04, Concordia University, Montréal, Québec, Canada, August 9-11, 2004*, pages 64–67, 2004. URL: https://www.cccg.ca/proceedings/2004/55.pdf.

[12] Erik D. Demaine, David Eppstein, Adam Hesterberg, Hiro Ito, Anna Lubiw, Ryuhei Uehara, and Yushi Uno. Folding a paper strip to minimize thickness. *J. Discrete Algorithms*, 36:18–26, 2016. doi:10.1016/j.jda.2015.09.003.

[13] Erik D. Demaine and Joseph S. B. Mitchell. Reaching folded states of a rectangular piece of paper. In *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pages 73–75, 2001. URL: https://erikdemaine.org/papers/PaperReachability_CCCG2001/.

[14] Linda Deneen and Gary Shute. Polygonizations of point sets in the plane. *Discrete Comput. Geom.*, 3(1):77–87, 1988. doi:10.1007/BF02187898.

[15] David Eppstein. Realization and connectivity of the graphs of origami flat foldings. *J. Comput. Geom.*, 10(1):257–280, 2019. doi:10.20382/jocg.v10i1a10.

[16] David Eppstein. Locked and unlocked smooth embeddings of surfaces. In *Proc. 34th Canadian Conference on Computational Geometry (CCCG 2022)*, pages 135–142, 2022.

[17] Martin Gardner. Flexagons: In which strips of paper are used to make hexagonal figures with unusual properties. *Scientific American*, 195(6):162–168, December 1956. doi:10.1038/scientificamerican1256-162.

[18] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

[19] Peter Jonsson, Victor Lagerkvist, Gustav Nordh, and Bruno Zanuttini. Complexity of SAT problems, clone theory and the exponential time hypothesis. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1264–1277. SIAM, 2013. doi:10.1137/1.9781611973105.92.

[20] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Comput. Sci.* Springer, 1994. doi:10.1007/BFb0045375.

[21] Thomas D. Morgan. Map Folding. Master's thesis, Massachusetts Institute of Technology, June 2012. URL: https://hdl.handle.net/1721.1/77030.

[22] Gaiane Panina and Ileana Streinu. Flattening single-vertex origami: The non-expansive case. *Computational Geometry*, 43(8):678–687, 2010. arXiv:1003.3490, doi:10.1016/j.comgeo.2010.04.002.

[23] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

[24] Ileana Streinu and Walter Whiteley. Single-vertex origami and spherical expansive motions. In *Discrete and Computational Geometry: Japanese Conference, JCDCG 2004, Tokyo, Japan, October 8-11, 2004, Revised Selected Papers*, volume 3742 of *Lecture Notes in Comput. Sci.*, pages 161–173. Springer, 2005. doi:10.1007/11589440_17.

A ETH-hardness

In our parameterized algorithm for flat folding, the dependence on ply comes from Lemma 5, which provides a time bound of $O(pw(p!)^{w+1})$ for computing the valid states of a single bag in a nice tree decomposition. The overall time bound is then this same bound, multiplied by the $O(n)$ bags of the decomposition. When $p = O(1)$, this bound reduces to single-exponential in w: the total time is $O(n2^{O(w)})$.

As we now show, a bound of this form is necessary under the exponential-time hypothesis [18], which for our purposes is most conveniently phrased as the assumption that there does not exist an algorithm for the 3SAT (satisfiability of 3-CNF Boolean formulae with n variables and m clauses) that has a sublinear running time bound of the form $2^{o(n+m)}$. Our proof uses NAE3SAT (not-all-equal-3-satisfiability), a variant of 3SAT in which there are again n Boolean variables, and in which certain triples of variables and their negations are not allowed to be equal. Standard NP-completeness reductions from 3SAT to NAE3SAT produce instances with $O(n + m)$ variables and clauses, from which it follows that under the exponential time hypothesis it is not possible to solve NAE3SAT instances in time subexponential in their numbers of variables or clauses. The same is known to be true more generally for a wide class of satisfiability-like problems including both 3SAT and NAE3SAT [19].

We base our hardness result on the proof by Bern and Hayes that flat foldability is NP-complete [5]. Bern and Hayes actually provide two proofs, one for unlabeled crease patterns and one for crease patterns labeled with mountain folds and valley folds, but both follow the same outline. They are reductions from NAE3SAT, and they produce crease patterns in the shape of a rectangle, where each variable of a NAE3SAT instance is represented by two closely spaced parallel zigzag paths of creases from the left side of the rectangle to the right side; none of these paths cross each other. Each clause of the NAE3SAT instance is represented by a small folded area near the top of the rectangle. Pairs of closely-spaced vertical fold lines connect the clauses to the variables, passing through the zigzag paths of variables that they do not interact with. Additional "noise" pairs of closely-spaced vertical fold lines are necessary to produce the zigzag pattern of the variable creases, but otherwise pass through the other variables without interacting with them. Each variable path, and each vertical pair of fold lines, have two locally-consistent folded states (used in the proof to represent the true and false truth assignment to each variable). The clause regions can only be flat-folded for truth assignments that satisfy the given clause. When a flat folding exists, and the construction is flat-folded, most of the paper has ply 1, with ply 3 along the folded regions near each variable gadget and vertical fold line, ply 5 at the points where two of these folded regions cross, and somewhat larger ply within the clause gadgets. Fig. 8 provides a schematic view of the crease patterns produced by these two reductions.

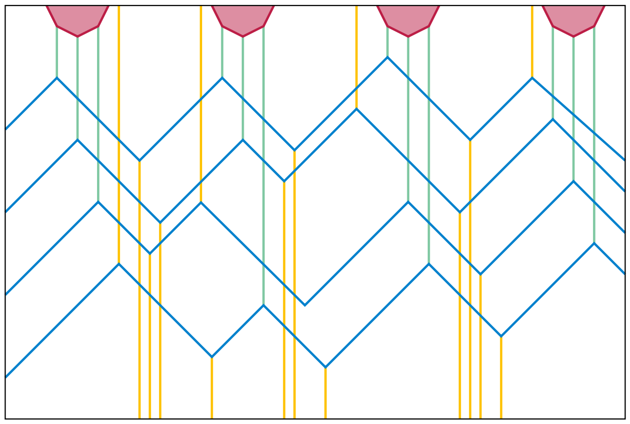


Figure 8: Schematic view of the crease patterns produced by the hardness reductions of Bern and Hayes [5]. The red regions at top are clause gadgets and the blue zigzag paths from left to right are variable gadgets. The variable gadgets are connected to the clause gadgets by vertical creases (light green) and additional "noise" vertical creases (yellow) connect to bends ("reflector gadgets") in the paths of the variable gadgets. Not shown: the additional reflectors needed to complement variables. Illustration modeled after Fig. 10 of Bern and Hayes.

Observation 7. *The local flat foldings of the crease patterns of Bern and Hayes have ply $O(1)$. For a NAE3SAT instance with n vertices and m clauses, they have treewidth $O(n)$, obtained by a path decomposition whose bags are the subsets of cells of the local flat folding intersected by vertical lines, in left-to-right order.*

Theorem 8. *If the exponential time hypothesis is true, it is not possible to test flat foldability of crease patterns of ply $O(1)$ and treewidth w in time $2^{o(w)}$, regardless of whether the pattern is labeled with mountain and valley folds or unlabeled.*

Proof. If such a fast test existed, then applying it to the crease patterns produced by the hardness reductions of Bern and Hayes would give an algorithm for NAE3SAT with time $2^{o(m)}$, contradicting the exponential time hypothesis. □

Piercing Unit Geodesic Disks[*]

Ahmad Biniaz[†] Prosenjit Bose[‡] Thomas Shermer[§]

Abstract

We prove that at most 3 points are always sufficient to pierce a set of m pairwise-intersecting unit geodesic disks inside a simple polygon P with n vertices of which n_r are reflex. We provide an $O(n + m \log n_r)$ time algorithm to compute these at most 3 piercing points. Our bound is tight since it is known that in certain cases, 3 points are necessary.

1 Introduction

The study of problems related to piercing a collection of convex sets has a rich history in Computational Geometry [10]. One of the most famous results in this area is Helly's theorem [15, 16] which states the following: Given n convex sets in $\Re^d$, with $n > d$, if every $d + 1$ convex sets have a nonempty intersection, then all n sets have a nonempty intersection. In other words, if a point pierces every $d + 1$ sets, then a point pierces all n sets. For Helly's theorem to hold, it is critical that every $d + 1$ sets have a point in common. Helly's theorem no longer holds if only d sets have a point in common. For example, given n lines in the plane, i.e. $d = 2$, where every pair of lines intersects but no three have a point in common, then $\Omega(n)$ points are required to pierce every line. On the other hand, given a set of n pairwise-intersecting disks in the plane, Danzer and Stachó independently showed that 4 points pierce all the disks [9, 22, 23]. Grünbaum [11] showed that 4 points are sometimes necessary thereby proving optimality. Neither the proof by Danzer nor the proof by Stachó lends itself to an efficient algorithm to actually compute these 4 points. From the computational perspective, Har-Peled et al. [14] presented a linear time algorithm to compute 5 points that pierce a set of pairwise-intersecting disks. Biniaz et al. [6] presented a simple linear time algorithm to find 5 piercing points using elementary geometric observations. Carmi et al. [8] presented a fairly involved linear time algorithm to compute 4 piercing points. In the case of a set of pairwise-intersecting unit disks, Hadwiger and Debrunner [13] showed that 3 points are sufficient to pierce the set. Biniaz et al. [6] showed that 3 points are sometimes necessary and presented a simple linear time algorithm to compute the piercing points. It is the fact that disks are *fat*, as opposed to lines which are *thin*, that allows a constant number of points to pierce pairwise-intersecting disks. This relationship between the number of points needed to pierce a family of planar pairwise-intersecting convex sets and the *fatness* of these sets has been explored in the literature [2, 5, 18, 20]. The most recent result we are aware of is by Bazarghani et al. [5] who show that $O(\alpha)$ points can pierce a set of pairwise-intersecting α-fat convex sets. Although there are several definitions of *fatness* in the literature, the definition that is used in [5] is the following: a convex set C is deemed α-fat if the ratio of the radius of the smallest disk that contains C and the largest disk that is contained in C is at most α.

In this paper, we focus on piercing problems in the geodesic setting. Specifically, we explore the following question: given a set of pairwise-intersecting geodesic disks inside a simple polygon, can a constant number of points pierce every disk? Given a simple polygon P, a geodesic disk centered on a point $x \in P$ is the set of points $y \in P$ such that the length of the shortest path from x to y in P is at most a constant r, the radius. This setting is more general than the setting in the Euclidean plane. In this setting, Bose et al. [7] showed that 14 points suffice to pierce a set of pairwise-intersecting geodesic disks inside a simple polygon and gave an $O(n+m \log n_r)$ time algorithm to compute these at most 14 piercing points where n is the number of vertices of P, n_r is the number of reflex vertices and m is the number of geodesic disks. Subsequently, Abu-Affash et al. [1] showed that 5 points suffice in this setting and provide an $O((n+m) \log n_r)$ time algorithm to find these 5 piercing points. This upper bound may not be tight since the best known lower bound on the number of points required to pierce a set of pairwise-intersecting geodesic disks is 4. Our main result is the following: we show that 3 points are always sufficient to pierce a set of pairwise-intersecting unit geodesic disks inside a simple polygon and provide an $O(n + m \log n_r)$ time algorithm to compute these 3 piercing points. Our bound is tight since the lower bound of 3 points in the plane also holds in the more general geodesic setting.

[*]Research support in part by NSERC.

[†]School of Computer Science, University of Windsor, abiniaz@uwindsor.ca

[‡]School of Computer Science, Carleton University, jit@scs.carleton.ca

[§]School of Computing Science, Simon Fraser University, shermer@sfu.ca

2 Notation and Preliminaries

Before presenting our main results, we first introduce some notation and preliminary lemmas. Let $P = v_0, \ldots, v_{n-1}$ be a simple n-vertex polygon. We use the convention that the interior of P lies to the right of the edge directed from v_i to v_{i+1}, i.e. the polygon is described in a clockwise fashion. In what follows, index manipulation is modulo the size of the set. In the case of the polygon, it is modulo n.

A segment between two points a, b is denoted as ab and its length is denoted as $|ab|$. Given two points $x, y \in P$, the geodesic (or shortest) path from x to y in P is denoted $\Pi(x, y)$. The length of this path, referred to as the *geodesic distance*, is the sum of the lengths of its edges and is denoted by $|\Pi(x, y)|$. The *geodesic metric* refers to P together with the geodesic distance function. A subset S of P is *geodesically convex* if, for all pairs of points $x, y \in S$, the geodesic path in P between x and y (i.e. $\Pi(x, y)$) is in S. Pollack et al. [21] proved the following lemma about distances between a point and a geodesic path.

Lemma 1 *[21] Let a, b, c be 3 distinct points in P. Define the function $g : \Pi(b, c) \to \Re$, as $g(x) = |\Pi(a, x)|$. Then g is a convex function with its maximum occurring either at b or c.*

Informally, a polygon P is *weakly simple* provided that a slight perturbation of the points on the boundary results in a simple polygon. See Akitaya et al. [4] for a formal definition of weakly-simple polygons as well as an algorithm to quickly recognize such polygons. A *pseudo-triangle* is a simple polygon with 3 convex vertices (the shaded region in Figure 1 is a pseudo-triangle). A *geodesic triangle* on points $a, b, c \in P$, denoted $\triangle(a, b, c)$, is a weakly-simple polygon whose boundary consists of $\Pi(a, b), \Pi(b, c)$ and $\Pi(c, a)$. In Figure 1, $\triangle(c_0, c_1, c_2)$ consists of the red paths and the shaded region. A *geodesic hexagon* is defined in a similar fashion but on six points in P. Let $X = \{x_0, x_1, \ldots, x_k\}$ be a set of at least 3 points in P. The set X is *geodesically collinear* if $\exists x_i, x_j \in X$ such that $X \subset \Pi(x_i, x_j)$. Given points a, b, and c in P that are not geodesically collinear, the shortest paths $\Pi(a, b)$ and $\Pi(a, c)$ follow a common path from a until they diverge at a point a' (note that a' could be a). Similarly, let b' be the point where $\Pi(b, a)$ and $\Pi(b, c)$ diverge, and c' be the point where $\Pi(c, a)$ and $\Pi(c, b)$ diverge. The geodesic triangle $\triangle(a', b', c')$ is simple (not weakly simple), has a', b', and c' as its convex vertices, and is a pseudo-triangle. We refer to $\triangle(a', b', c')$ as the *geodesic core* of $\triangle(a, b, c)$ and denote it as $\triangledown(a, b, c)$; the shaded region in Figure 1 is the geodesic core of $\triangle(c_0, c_1, c_2)$. These properties were also observed in Pollack et al. [21].

This leads to a natural generalization of the notions of orientation, angles, and sidedness for geodesics. Given two distinct points $a, b \in P$, the *orientation* of a point a with respect to b in P is the counter-clockwise angle that the first edge of $\Pi(a, b)$ makes with the positive x-axis. Orientations are between 0 (inclusive) and 2π (exclusive). Given 3 points $a, b, c \in P$ that are not geodesically collinear, we denote by $\angle abc$ the convex angle at b' in the geodesic core $\triangledown(a, b, c)$. When a, b, c are geodesically collinear then $\angle abc$ is π if $b \in \Pi(a, c)$, and 0 otherwise. We say that b is to the left of $\Pi(a, c)$ if the convex vertices in $\triangledown(a, b, c)$ appear in the order a', b', c' when traversing the boundary in clockwise order starting at a'; otherwise, b is to the right. When referring to points of P to the left or right of an edge ab of P, we consider ab to be $\Pi(a, b)$.

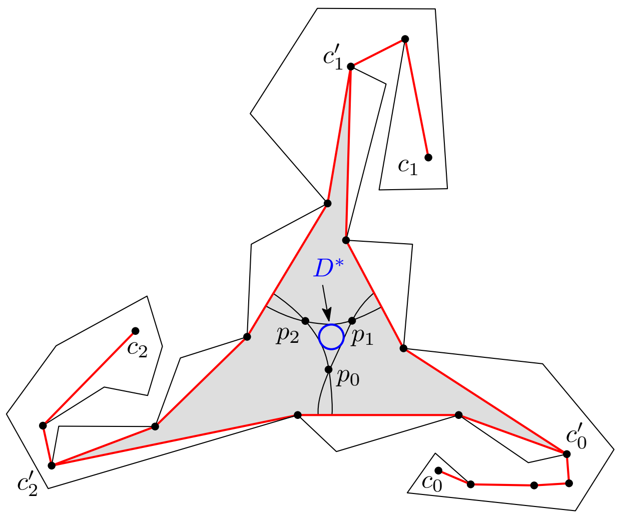


Figure 1: Basic definitions.

A *geodesic disk* centered at $c \in P$ with radius $r \geq 0$ is the set $\{y \in P : |\Pi(c, y)| \leq r\}$. A geodesic disk is geodesically convex and its boundary may be composed of several arcs of different curvature [21]. Two geodesic disks are *tangent* when the geodesic distance between the centers of the disks is exactly the sum of the radii. A *unit* geodesic disk is a geodesic disk with radius 1.

3 Upper bound on number of piercing points

In this section, we prove that 3 points suffice to pierce any set of pairwise-intersecting geodesic unit disks. Throughout this paper, we will be working with a collection $\mathcal{D} = \{D_0, D_1, \ldots, D_{m-1}\}$ of pairwise-intersecting unit geodesic disks whose respective centers $c_0, c_1, \ldots c_{m-1}$ are in P. We define D^* as the smallest geodesic disk that intersects each member of $\mathcal{D}$, with c^* and r^* being the center and radius of D^*, respectively. The set $\mathcal{D}$ is called *Helly* if there is one point that pierces all the disks. Every disk in $\mathcal{D}$, by definition, intersects D^*. We use D^* to compute the 3 points that suffice to pierce $\mathcal{D}$, when $\mathcal{D}$ is not Helly. The following lemma about properties of D^* when $\mathcal{D}$ is not Helly, proven in [7], will be useful in the sequel.

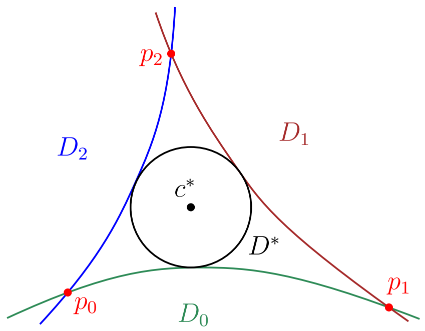


Figure 2: Close-up of p_0, p_1, p_2.

Lemma 2 *[7] If $\mathcal{D}$ is* **not** *Helly, then the disk D^* has the following properties:*

1. *the radius $r^* > 0$, where r^* is the radius of D^*,*
2. *D^* is tangent to at least 3 geodesic disks D_0, D_1, D_2 in $\mathcal{D}$ at 3 distinct points t_0, t_1 and t_2, respectively,*
3. *D^* does not intersect the boundary of the geodesic core $\nabla(c_0, c_1, c_2)$, where c_i is the center of disk D_i, for $i \in \{0, 1, 2\}$,*
4. *The boundary of D^* is a circle,*
5. *c^* is contained in the interior of $\triangle(t_0, t_1, t_2)$.*

The properties of D^* that are important to note are the following. First, even though D^* is a geodesic disk in P, its boundary is actually a circle that does not intersect the boundary of P; see Figure 1. Second, the fact that $\mathcal{D}$ consists of pairwise-intersecting unit disks implies that D^* must be tangent to 3 disks in $\mathcal{D}$ as opposed to 2, which can be the case when the disks are not pairwise-intersecting. In the remainder of the paper, we use the notation in Lemma 2 to refer to the three disks tangent to D^*, their tangency points, and centers. We begin by giving an upper bound on the radius r^* of D^*.

Lemma 3 *The radius r^* of D^* is at most $(2/\sqrt{3}) - 1$.*

Proof. If $\mathcal{D}$ is Helly, then $r^* = 0$, thus, we only need to consider the case when $\mathcal{D}$ is not Helly. Since $\sum_{i=0}^{2} \angle c_i c^* c_{i+1} = 2\pi$, we can assume without loss of generality that $\angle c_1 c^* c_2 \geq 2\pi/3$. Denote by ray$(a, b)$ the half-line with initial point a containing b. Let $c^* b_1$ be the first edge of $\Pi(c^*, c_1)$, as in Figure 4. Define b_1' as the first point along ray(c^*, b_1) where it intersects with $\Pi(c_1, c_2)$. This intersection must exist by the Jordan Curve Theorem [24] since c^* is inside $\nabla(c_0 c_1 c_2)$. Note that it may be the case that b_1' is b_1. Let c_1' be the point on ray(c^*, b_1') such that $|c^* c_1'| = |\Pi(c^*, c_1)|$. Define b_2' and c_2' analogously. The segment $c^* c_1'$ can be viewed as an *unfolding* of $\Pi(c^*, c_1)$ onto ray(c^*, b_1'). Thus, since D^* and D_1 are tangent, we have that $|\Pi(c^*, c_1)| = |c^* c_1'| = |c^* b_1'| + |b_1' c_1'| = 1 + r^*$. Similarly, $|\Pi(c^*, c_2)| = |c^* b_2'| + |b_2' c_2'| = 1 + r^*$. Since $\angle c_1' c^* c_2' \geq 2\pi/3$, by the cosine law, we have that $|c_1' c_2'| \geq \sqrt{3}(1 + r^*)$.

By the triangle inequality of the geodesic metric, $|\Pi(c^*, c_1)| \leq |c^* b_1'| + |\Pi(b_1', c_1)|$. Since $|\Pi(c^*, c_1)| = |c^* b_1'| + |b_1' c_1'|$, we have that $|b_1' c_1'| \leq |\Pi(b_1', c_1)|$. By the same argument, $|b_2' c_2'| \leq |\Pi(b_2', c_2)|$. Therefore, we have that $|\Pi(c_1, c_2)| = |\Pi(c_1, b_1')| + |\Pi(b_1', b_2')| + |\Pi(b_2', c_2)| \geq |c_1' b_1'| + |\Pi(b_1', b_2')| + |b_2' c_2'| \geq |c_1' c_2'|$.

Since D_1 and D_2 have unit radius and intersect, we have that $2 \geq |\Pi(c_1, c_2)| \geq |c_1' c_2'| \geq \sqrt{3}(1 + r^*)$. We conclude that $r^* \leq (2/\sqrt{3}) - 1$.

□

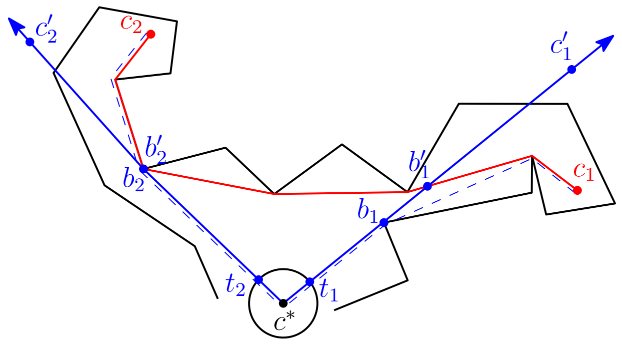


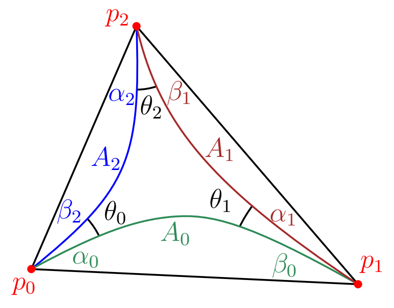
Figure 4: Illustration of the proof of Lemma 3.

For $i \in \{0, 1, 2\}$, let p_i be the point of $D_i \cap D_{i-1}$ closest to c^* (Figure 2). These points must exist because the disks in $\mathcal{D}$ are pairwise-intersecting. Moreover, in our main theorem, we will prove that these three points pierce the set $\mathcal{D}$.

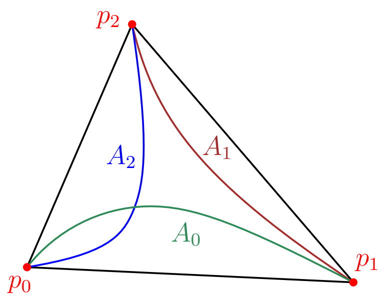
Lemma 4 *The points p_0, p_1 and p_2 are in the geodesic core $\nabla(c_0, c_1, c_2)$.*

Proof. We show that $p_2 \in \nabla(c_0, c_1, c_2)$. The same argument shows that both p_1 and p_0 are in $\nabla(c_0, c_1, c_2)$. Consider $\triangle(b_1', b_2', c^*)$ where b_1' and b_2' are defined as in the proof of Lemma 3 and illustrated in Figure 4. Recall that $|\Pi(c^*, c_1)| = 1 + r^*$ since D_1 is tangent to D^*. By construction, we have that $|\Pi(c^*, c_1)| = |c^* b_1'| + |b_1' c_1'|$. Since $|c^* b_1'| > r^*$, we have that $|b_1' c_1'| = |\Pi(c_1, b_1')| < 1$. Note that by construction of b_1', we have that $\Pi(c_1, c_2) = \Pi(c_1, b_1') + \Pi(b_1', c_2)$. Given that $|\Pi(c_1, b_1')| < 1$, we have that the boundary of D_1 intersects $\Pi(c_1, c_2)$ at a point x on $\Pi(b_1', c_2)$. Similarly, the boundary of D_2 intersects $\Pi(c_1, c_2)$ at a point y on $\Pi(b_2', c_1)$.

By construction, we have that c^* is a convex vertex of the geodesic triangle $\triangle(b_1', b_2', c^*)$. Since D_1 and D_2



(a)



(b)

Figure 3: Points, arcs, and angles.

intersect, we have that $|\Pi(c_1, c_2)| \leq 2$. If $|\Pi(c_1, c_2)| = 2$, in other words, the point x and y coincide, then p_2 is on $\Pi(c_1, c_2)$ and therefore $p_2 \in \nabla(c_0, c_1, c_2)$. Otherwise, we consider the case when $|\Pi(c_1, c_2)| < 2$. In this case, notice that as we traverse $\Pi(c_1, c_2)$ from c_1 to c_2, we must encounter y before x.

Consider the arc B_1 to be the portion of the boundary of D_1 from t_1, the point of tangency between D_1 and D^*, to x. Since this arc at t_1 enters $\triangle(b'_1, b'_2, c^*)$, by the Jordan curve theorem [24], it intersects either $\Pi(b'_1, b'_2)$ or the segment $c^*b'_2$. Let us consider the latter case first. Assume that B_1 intersects $c^*b'_2$ at a point z. Let B'_1 be the portion of B_1 from t_1 to z. Consider the closed region R consisting of the segment zc^*, the segment c^*t_1 and B'_1. We now define the arc B_2 to be the portion of the boundary of D_2 from t_2 to y. At t_2, the arc B_2 enters the region R. Since y is outside of R, by the Jordan curve theorem, B_2 must intersect the boundary of R. This intersection point, which is p_2, must be on B'_1 since B_2 cannot intersect c^*t_1 as every point on that segment is farther than 1 from c_2. Thus, p_2 is in $\triangle(b'_1, b'_2, c^*)$ since B'_1 is.

For the case where B_1 intersects $\Pi(b'_1, b'_2)$, we use the same argument except that the boundary of the region R consists of B_1, $\Pi(x, b'_2)$, b'_2c^* and c^*t_1. Since we encounter y before x when we traverse $\Pi(c_1, c_2)$ from c_1 to c_2, the point y is outside R. Thus B_2 must intersect the boundary of R, and similar to previous case this intersection which is p_2 must be through B_1 in the triangle $\triangle(b'_1, b'_2, c^*)$. Therefore, we have that $p_2 \in \nabla(c_0, c_1, c_2)$. □

By the proof of Lemma 4, p_2 lies in $\triangle(b'_1, b'_2, c^*)$ which is essentially a star shaped polygon with center c^*. Thus the segment c^*p_2 lies in $\triangle(b'_1, b'_2, c^*)$ which is a subset of $\nabla(c_0, c_1, c_2)$. Applying a similar argument to p_0 and p_1 we have the following corollaries.

Corollary 5 *The line segment c^*p_i is in $\nabla(c_0, c_1, c_2)$.*

Recall c'_0, c'_1, and c'_2 as the convex vertices of the geodesic core $\nabla(c_0, c_1, c_2)$.

Corollary 6 *The geodesic hexagon $c'_0p_1c'_1p_2c'_2p_0$ is a subset of the geodesic triangle $\triangle c_0c_1c_2$.*

Refer to Figure 3(a) for the following. For $i \in \{0, 1, 2\}$, let A_i be the arc on the boundary of D_i from p_i to p_{i+1}. Let θ_i be the clockwise angle from A_{i-1} to A_i at p_i. If $\theta_i = 0$ then the disks D_{i-1} and D_i are tangent at p_i. If $\theta_i > 0$ then D_{i-1} and D_i have a positive area of overlap, starting at p_i. The case when $\theta_i < 0$ cannot happen since p_i is the intersection point closest to c^*. Note this in Figure 3(b) where p_0 should be at the other intersection of arcs A_0 and A_2.

For $i \in \{0, 1, 2\}$, let α_i be the angle from A_i to the line segment p_ip_{i+1} at p_i, and β_i be the angle from A_i to the line segment p_ip_{i+1} at p_{i+1}; see Figure 3(a).

Lemma 7 *For $i \in \{0, 1, 2\}$, $|p_ip_{i+1}| \leq 1$.*

Proof. Consider a parameter s that denotes the distance we have moved as we move from p_i to p_{i+1} along A_i. The coordinates of a point $x \in A_i$ as well as the tangent t to A_i at point x can be expressed as a function of this parameter s. See Figure 5, where the tangents are shown as red arrows. Let Δt denote the change in angle of this tangent from p_i to p_{i+1}. Then $\Delta t = \alpha_i + \beta_i$. This can be seen in the figure, letting q be the point where the tangent is parallel to the segment p_ip_{i+1}. Then the tangent sweeps out α_i as it moves from p_i to q, and then sweeps out β_i as it moves from q to p_i.

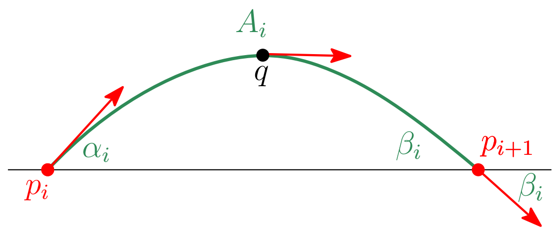


Figure 5: Tangents to A_i.

Let $\kappa(s)$ denote the curvature of A_i with respect to parameter s. Then, by definition of the integral of curvature taken along A_i, we have that $\Delta t = \int_{A_i} \kappa(s) ds$.

Since D_i is a unit geodesic disk, it has curvature at least 1 on all of its boundary arcs. This is because every boundary arc of D_i comes from a circle whose radius is at most 1. Since $\kappa(s) \geq 1$, we have $\Delta t \geq \int_{A_i} 1 ds$. But the latter integral is simply the length of the arc A_i. Since $\alpha_i + \beta_i = \Delta t$, we have that $\alpha_i + \beta_i \geq |A_i|$.

Because of the lower bound of 1 on the curvature, the length of A_i will be at least as large as the length of a (uniformly) curvature-1 curve from p_i to p_{i+1} This uniform curve is a circular arc A'_i of radius 1 with some center which we denote as c'_i; see Figure 6. Denote by C'_i the unit circle centered at c'_i. We have $A'_i \subset C'_i$.

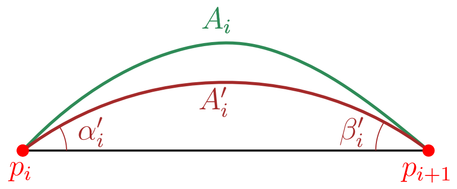


Figure 6: A curvature-1 curve A'_i.

Claim 1 *For $i \in \{0,1,2\}, |p_ip_{i+1}|$ is maximized when C'_0, C'_1, and C'_2 are pairwise tangent.*

Proof. By definition, C'_i and C'_{i+1} have a non-empty intersection. Define L'_i as the line through c'_i and c'_{i+1}. For sake of a contradiction, we first consider the case where none of the disks are tangent to each other but $|p_ip_{i+1}|$ is maximized. Move c'_0 in the direction perpendicular to L'_1 away from L'_1 until C'_0 becomes tangent to either C'_1 or C'_2. During this process, p_2 remains fixed and p_0p_2, p_1p_2, p_0p_1 increase in length, which is a contradiction. Now, without loss of generality, assume that only C'_0 and C'_1 are tangent. By moving c'_2 in the direction perpendicular to L'_0 away from L'_0 until C'_2 becomes tangent to either C'_0 or C'_1, once again, p_1 remained fixed and p_0p_1, p_1p_2, p_0p_2 increase in length, which is a contradiction. Finally, without loss of generality, assume that only C'_0 and C'_2 are not tangent. Rotate C'_0 around c'_1, while keeping it tangent to C'_1, until C'_0 is tangent to C'_2. Here we note that p_2 remains fixed, and p_0p_2, p_1p_2, p_0p_1 increase in length. Therefore, we conclude that each $|p_ip_{i+1}|$ is maximized when C'_0, C'_1, and C'_2 are pairwise tangent. This finishes our proof of Claim 1.

By Claim 1, each $|p_ip_{i+1}|$ is maximized when C'_0, C'_1, and C'_2 are pairwise tangent, in which case $\triangle(p_0, p_1, p_2)$ must be an equilateral triangle with side length 1.

□

Corollary 8 *For $i \in \{0,1,2\}$,*

$$|c^*p_i| \leq \sqrt{r^*(2+r^*)} \leq 0.578.$$

Proof. Using the same transformation as in the proof of Claim 1, we can see that for $i \in \{0,1,2\}$, $|c^*p_i|$ is maximized when the circles C'_i are pairwise tangent and the points p_0, p_1, p_2 form an equilateral triangle. This means that c'_i, c^* and p_i form a right triangle with side lengths $1, 1+r^*$ and $|c^*p_i|$. Pythagoras' theorem gives the bound on $|c^*p_i|$ and the numerical upper bound we get from the upper bound on r^* in Lemma 3. □

Theorem 9 *Let $\mathcal{D}$ be a collection of pairwise-intersecting unit geodesic disks inside a simple polygon P. Then there are three points inside P such that each disk of $\mathcal{D}$ contains at least one of the points.*

Proof. Let D^+ be the radius-$1+r^*$ geodesic disk centered at c^*, and C^+ be the geodesic circle that is the boundary of D^+. The circle C^+ contains arcs at distance $1+r^*$ from c^* and segments of the boundary of P at distances less than that. If we extend the line segment c^*p_i in a straight line from p_i, we will hit C^+ at some point q_i (which could be the same as p_i). The c_i's (the centers of the three disks tangent to D^*) and q_i's divide the circle C^+ into six sections; we concentrate on the section between c_1 and q_1; a symmetric argument applies to the other five sections.

Since both ends of $\Pi(c_1, c_0)$ are at geodesic distance $1+r^*$ from c^*, any point on $\Pi(c_1, c_0)$ is at distance no more than $1+r^*$ from c^* (by Lemma 1). This implies that the arcs of C^+ (which are at distance $1+r^*$ from c^*) do not intersect the interior of the geodesic core of the geodesic triangle $\triangle c_0c_1c_2$. Since there is no boundary of P in the interior of any geodesic core, the segments of C^+ also do not intersect the interior of the geodesic core of $\triangle c_0c_1c_2$. Because this is true for all six sections of C^+, C^+ does not intersect the interior of the geodesic core.

Let c_T be a point on C^+ non-strictly between c_1 and q_1. Because c_T is not in the interior of the geodesic core of $\triangle c_0c_1c_2$, $\Pi(c_T, c^*)$ intersects $\Pi(c_1, c_0)$. This implies that $\Pi(c_T, c^*)$ also intersects $\Pi(c_1, p1)$, as the geodesic hexagon $c'_0p_1c'_1p_2c'_2p_0$ (which contains c^*) must be inside the geodesic core of $\triangle c_0c_1c_2$, by Corollary 6. Let u be the intersection point of $\Pi(c_T, c^*)$ and $\Pi(c_1, p1)$, and let t_T be the point where $\Pi(c_T, c^*)$ crosses the boundary of D^*. See Figure 7.

The distance $d(c_1, p_1)$ is equal to $d(c_1, u) + d(u, p_1) = 1$ since p_1 is on the boundary of D_1. The distance $d(c_1, u) + d(u, t_T) \geq 1$, since D_1 is tangent to D^*. So $d(u, t_T) \geq d(u, p_1)$ and therefore $d(c_T, u) + d(u, t_T) \geq d(c_T, u) + d(u, p_1)$. The left-hand side of that last inequality is simply 1, and the right-hand side is an upper bound on the distance $d(c_T, p_1)$, so we get $1 \geq d(c_T, p_1)$, or that p_1 pierces the disk of radius one centered at c_T.

Now consider a unit disk D in our collection of disks $\mathcal{D}$. The center c of D lies inside the radius $1+r^*$ disk around c^*, and without loss of generality, it lies in a

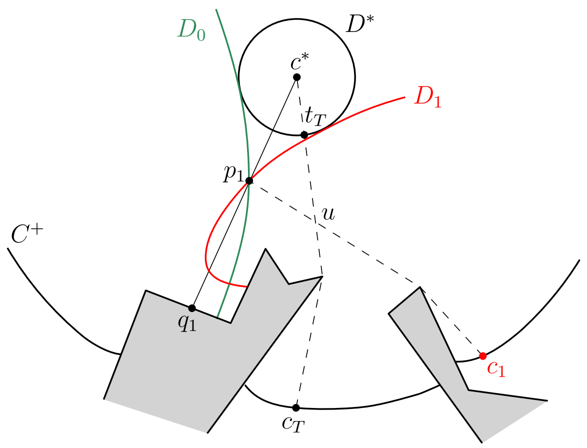


Figure 7: $\Pi(c_T, c^*)$ intersects $\Pi(c_1, p_1)$ at u.

direction between c_1 and p_1 from c^*. We extend the last segment of $\Pi(c^*, c)$ until it reaches the radius $1+r^*$ circle at a point c_T. The center c lies on $\Pi(c_T, c^*)$, the distance $d(c_T, p_1) \leq 1$ as discussed above, and the distance $d(c^*, p_1) \leq 1$ (by Corollary 8). Thus $d(c, p_1) \leq 1$ by Lemma 1, and hence p_1 pierces D.

Therefore, the three points p_0, p_1, and p_2 pierce the entire collection $\mathcal{D}$. □

4 Algorithmic Considerations

In this section, we describe an algorithm to compute the piercing points. The input to the algorithm is $\mathcal{D}$. First, compute D^* in $O(n+m \log n_r)$ time using the algorithm described in [7]. This is achieved since it was shown in [7] that computing D^* is an LP-type problem.

The reason that the run-time has a $\log n_r$ term as opposed to a $\log n$ term is that given a polygon P, we first apply a geodesic-preserving simplification of P in $O(n)$ time to get a polygon $P' \supset P$ of size $O(n_r)$ where n_r is the number of reflex vertices in P, such that the shortest path from x to y in P is identical to the shortest path from x to y in P' [3]. Then, we preprocess P' in $O(n_r)$ time to answer in $O(\log n_r)$ time the length of the shortest path from x to y and $O(\log n_r + k)$ to report the k edges on the shortest path [12, 17]. With these tools in hand, Bose et al. [7] apply Matousek et al.'s [19] general framework for solving LP-type problems to find D^* within the stated amount of time.

If $r^* = 0$, then c^* is returned as the point that pierces $\mathcal{D}$. If $r^* > 0$, then in $O(n)$ time, compute $\nabla(c_0c_1c_2)$ with 3 queries to the shortest path data structure constructed above. Now all that remains is to compute p_0, p_1 and p_2. We show how to compute p_0 in $O(n)$ time. The other two points are computed in a similar manner. Recall t_i as the point of tangency between D^* and D_i. To compute p_0, we need to intersect the arc A_0 with the arc A_2. Each arc A_i consists of at most n_r pieces of circular arcs inside $\nabla(c_0c_1c_2)$. Essentially, to find p_0, we walk along A_0 from t_0 towards $\Pi(c_0, c_2)$, and along A_2 from t_2 towards $\Pi(c_0, c_2)$. By always advancing on the arc that is furthest away from $\Pi(c_0, c_2)$, we eventually find p_0 in $O(n)$ time.

The cost of finding p_0, p_1, p_2 is dominated by the cost of finding D^*. We conclude with the following:

Theorem 10 *Given a set $\mathcal{D}$ of m pairwise-intersecting disks in a simple polygon P on n vertices and n_r reflex vertices, we can compute the at most 3 points that pierce $\mathcal{D}$ in $O(n + m \log n_r)$ time.*

5 Conclusion

Theorem 10 settles the question of how many points are sufficient to pierce a set of pairwise-intersecting unit disks in the geodesic setting. It would be interesting to prove that the runtime of our algorithm is optimal. We leave as an open question to determine whether 4 or 5 points are necessary to pierce pairwise-intersecting geodesic disks of arbitrary radius. When the radii are arbitrary, 4 points are sometimes necessary and always sufficient in the Euclidean setting. In the geodesic setting, the best known lower bound is 4 (from the lower bound example in the Euclidean setting) and the upper bound is 5 piercing points [1]. It would be interesting to close this gap.

References

[1] A. K. Abu-Affash, P. Carmi, and M. Maman. Piercing pairwise intersecting geodesic disks by five points. *Comput. Geom.*, 109:101947, 2023.

[2] P. K. Agarwal, M. J. Katz, and M. Sharir. Computing depth orders for fat objects and related problems. *Computational Geometry: Theory and Applications*, 5(4):187–206, 1995.

[3] O. Aichholzer, T. Hackl, M. Korman, A. Pilz, and B. Vogtenhuber. Geodesic-preserving polygon simplification. *Int. J. Comput. Geometry Appl.*, 24(4):307–324, 2014.

[4] H. A. Akitaya, G. Aloupis, J. Erickson, and C. D. Tóth. Recognizing weakly simple polygons. *Discret. Comput. Geom.*, 58(4):785–821, 2017.

[5] S. Bazargani, A. Biniaz, and P. Bose. Piercing pairwise intersecting convex shapes in the plane. In *LATIN*, volume 13568 of *Lecture Notes in Computer Science*, pages 679–695. Springer, 2022.

[6] A. Biniaz, P. Bose, and Y. Wang. Simple linear time algorithms for piercing pairwise intersecting disks. In *CCCG*, pages 228–236, 2021 (to appear in CGTA).

[7] P. Bose, P. Carmi, and T. C. Shermer. Piercing pairwise intersecting geodesic disks. *Comput. Geom.*, 98:101774, 2021.

[8] P. Carmi, M. J. Katz, and P. Morin. Stabbing pairwise intersecting disks by four points. *CoRR*, abs/1812.06907, 2018.

[9] L. Danzer. Zur Lösung des Gallaischen Problems über Kreisscheiben in der Euklidischen Ebene. *Studia Scientiarum Mathematicarum Hungarica*, 21(1-2):111–134, 1986.

[10] J. E. Goodman, J. O'Rourke, and C. Toth, editors. *Handbook of Discrete and Computational Geometry, Third Edition*. Chapman and Hall/CRC, 2017.

[11] B. Grünbaum. On intersections of similar sets. *Portugal. Math.*, 18:155–164, 1959.

[12] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989.

[13] H. Hadwiger and H. Debrunner. Ausgewählte Einzelprobleme der kombinatorischen Geometrie in der Ebene. *Enseignement Math. (2)*, 1:56–89, 1955.

[14] S. Har-Peled, H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, M. Sharir, and M. Willert. Stabbing pairwise intersecting disks by five points. *Discret. Math.*, 344(7):112403, 2021.

[15] E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.

[16] E. Helly. Über Systeme von abgeschlossenen Mengen mit gemeinschaftlichen Punkten. *Monatshefte für Mathematik*, 37(1):281–302, 1930.

[17] J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Inf. Process. Lett.*, 38(5):231–235, 1991.

[18] M. J. Katz. 3-d vertical ray shooting and 2-d point enclosure, range searching, and arc shooting amidst convex fat objects. *Computational Geometry*, 8(6):299–316, 1997.

[19] J. Matousek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.

[20] F. Nielsen. On point covers of c-oriented polygons. *Theo. Comp. Sci.*, 265(1–2):17–29, 2001.

[21] R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4:611–626, 1989.

[22] L. Stachó. Über ein Problem für Kreisscheibenfamilien. *Acta Scientiarum Mathematicarum (Szeged)*, 26:273–282, 1965.

[23] L. Stachó. A gallai-féle körletuzési probléma megoldása. *Mat. Lapok*, 32(1-3):19–47, 1981-84.

[24] C. Thomassen. The jordan–schönflies theorem and the classification of surfaces. *American Mathematical Monthly*, 99(2):116–130, 1992.

Super Guarding and Dark Rays in Art Galleries

MIT CompGeom Group* Hugo A. Akitaya† Erik D. Demaine‡ Adam Hesterberg§ Anna Lubiw¶
Jayson Lynch‖ Joseph O'Rourke** Frederick Stock††

Abstract

We explore an Art Gallery variant where each point of a polygon must be seen by k guards, and guards cannot see through other guards. Surprisingly, even covering convex polygons under this variant is not straightforward. For example, covering every point in a triangle $k{=}4$ times (a ***4-cover***) requires 5 guards, and achieving a 10-cover requires 12 guards. Our main result is tight bounds on k-covering a convex polygon of n vertices, for all k and n. The proofs of both upper and lower bounds are nontrivial. We also obtain bounds for simple polygons, leaving tight bounds an open problem.

1 Introduction

The original Art Gallery Theorem showed that $\lfloor n/3 \rfloor$ guards are sometimes necessary and always sufficient to guard a simple polygon P of n vertices [O'R87]. (Throughout, P includes its boundary ∂P.) There have been many interesting variants explored since then. In this paper we explore two variants that are interesting in combination, although not individually.

(1) *Guards blocking guards*: Suppose guards cannot see through other guards.[1] More precisely, if g_1 and g_2 are guards, and g_1, g_2, p are on a line in that order, then point p is not visible from g_1. Still the original bound $\lfloor n/3 \rfloor$ holds, because g_2 can continue g_1's line-of-sight to p, picking it up where that line-of-sight terminates at g_2.

(2) *Multiple coverage*: Suppose every point in the closed polygon must be seen by k guards i.e., the guards must ***k-cover*** the polygon. The problem of k-guarding has been explored under various restrictions on guard location [BBC+94, Sal09, BEK13]. If multiple guards can be co-located at the same point, then this is trivial. If co-location is disallowed, but guards can see through other guards, then this still reduces to the case $k = 1$ since we can replace a single guard by a cluster of k guards. (We detail the argument in Section 4.)

So neither of these variations is "interesting" by itself in the sense that easy arguments lead to $\lfloor n/3 \rfloor$ bounds. However, consider now mixing these two variants:

> **Q**: How many guards are necessary and sufficient to cover a simple polygon P of n vertices so that every point of P is seen by at least k guards, where guards cannot be co-located, and each guard blocks lines-of-sight through it?

To our surprise, answering **Q** is not straightforward, even for convex polygons, even for triangles. For example, to cover a triangle to depth $k = 3$, one guard at each vertex suffices. Note here we consider a guard to see itself. But to cover to depth $k = 4$ requires $g = 5$ guards; see Fig. 9. And covering to depth $k = 10$ requires $g = 12$ guards.

The main result of this paper is the following theorem. We use n for the number of vertices, k for the depth of cover, and g for the number of guards.

Theorem 1 *For a closed convex n-gon, coverage to depth k requires $g \in \{k, k+1, k+2\}$ guards:*

(1) *For $k \le n$: $g = k$ guards are necessary and sufficient.*

(2) *For $n < k < 4n-2$: $g = k+1$ guards are necessary and sufficient.*

(3) *For $4n - 2 \le k$: $g = k + 2$ guards are necessary and sufficient.*

Thus there are three regimes depending on the relationship between n and k. For triangles, $n = 3$, the following table details those regimes:

k	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
g	1	2	3	5	6	7	8	9	10	12	13	$\cdots$

*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 CompGeom Group" (without "et al.").

†U. Mass. Lowell, hugo_akitaya@uml.edu

‡MIT, edemaine@mit.edu

§Harvard U., achesterberg@gmail.com

¶U. Waterloo, alubiw@uwaterloo.ca

‖MIT, jaysonl@mit.edu

**Smith College, jorourke@smith.edu

††U. Mass. Lowell, fbs9594@rit.edu

[1]This was posed as an exercise in [DO11], Exercise 1.28, p. 14.

Another example: For $n = 4$, $g = 14$ guards 13-cover, but a 14-cover requires $g = 16$ guards. See ahead to Fig. 10.

Our primary focus is proving Theorem 1. We also obtain in Lemma 8 tight bounds for a convex wedge, which can be viewed as a 2-sided unbounded convex polygon. Finally, we briefly address simple polygons in Theorem 7, which we do not consider as natural a fit as the question for convex polygons.

1.1 Dark Rays and Dark Points

With some abuse of notation, we will identify both a guard and that guard's location as g_i. Let g_1 and g_2 be two guards visible to one another. We say that g_2 ***generates*** a ***dark ray at*** g_1, which is a ray contained in the line through g_1 and g_2, incident to and rooted at g_1 and invisible to g_2. And similarly, g_1 generates a dark ray at g_2.

A point is called ***dark*** if it is contained in a dark ray, and ***d-dark*** if it is contained in at least d dark rays.

Because a d-dark point is hidden from d guards, we obtain an immediate relationship between dark rays and multiple guarding for convex polygons.

Observation 1

(1) *k-guarding with $g = k$ guards is possible if and only if there is no dark point inside P, i.e., all dark rays are strictly exterior to P.*

(2) *k-guarding with $g = k + 1$ guards is possible if and only if there is no 2-dark point inside P.*

(3) *k-guarding with $g = k + 2$ guards is always possible because we can perturb the guards to avoid 3-dark points, as justified in Appendix A.4.*

1.2 Outline of Proof of Theorem 1

Most steps of the proof follow directly from Observation 1, with the exception of the following non-trivial result.

Theorem 2 *The maximum number of guards that can be placed in a convex n-gon P without creating 2-dark points in P is $4n - 2$.*

We prove the upper bound (at most $4n - 2$ guards) in Section 2 and the lower bound ($4n - 2$ is possible) by a direct construction in Section 3. Both directions are non-trivial, and their proofs constitute the main focus of the paper. Assuming these results, the proof of Theorem 1 proceeds as follows:

To k-cover when $k \leq n$ (regime (1)) it is clear that k guards are necessary. For sufficiency, place k guards at vertices of polygon P. Then all dark rays are exterior to P, so by Observation 1(1), this is a k-cover.

To k-cover when $n < k < 4n - 2$ (regime (2)) the necessity of $k + 1$ guards follows from Lemma 9 (Appendix A.2) where we show that any placement of $n+1$ guards in a convex P results in a dark point inside P. Sufficiency is proved by Observation 1(2) (that we only need to avoid 2-dark points) and the lower bound of Theorem 2 (that we can place $k+1$ points without creating 2-dark points), since $k + 1 \leq 4n - 2$.

To k-cover when $4n - 2 \leq k$ (regime (3)) the sufficiency of $k + 2$ guards follows from Observation 1(3). Necessity is proved by the upper bound of Theorem 2.

2 $4n - 2$ Upper Bound

In this section we prove that at most $4n - 2$ guards can be placed in a convex n-gon P without creating 2-dark points in P.

2.1 Triangle Lemma

The following lemma is a key tool in the proof of the upper bound. It establishes that, excluding the exceptional case, any triangle of guards in P may only contain one additional guard if we are to avoid 2-dark points in T.

Lemma 3 (Triangle) *Suppose some guards are placed in P without creating 2-dark points. Let T be a closed triangle in P with guards g_1, g_2, g_3 at its corners. Then, with one exception, T contains at most one more guard.*

The exceptional case allows two guards, g_4, g_5, in T when (up to relabelling) g_1g_3 is an edge of P, g_4 lies on that edge, and g_2, g_5, g_4 are collinear.

Proof. Refer to Fig. 1(a,b) throughout. We first discuss the non-exceptional case. First suppose that there is an extra guard g_4 strictly interior to T. Then g_1, g_2, g_3 generate 3 dark rays at g_4, each of which crosses a different edge of T. The same would be true for a second strictly interior guard g_5. So a dark ray at g_5 must cross a dark ray at g_4 to reach an edge of T. The result is a 2-dark point, marked x in (a) of the figure. Since we assumed no 2-dark points in P, there cannot be two extra guards interior to T.

Suppose now that g_4 lies on edge $e = g_1g_3$ of T. Then left and right of g_4 on e are dark rays generated by g_1 and g_3. Placing g_5 at any point not collinear with g_4 and g_2 leads to a dark ray at g_5, generated by g_2, crossing e to form a 2-dark point there.

We are left with the exceptional case, illustrated in (b) of the figure: g_4 lies on an edge of T, and g_5 is collinear with g_4 and the opposite corner of the triangle, g_2 in the case illustrated. There are no 2-dark points inside T. The dark ray at g_5 generated by g_2 contains the dark ray at g_4 generated by g_5 so, to avoid 2-dark points inside P, g_4 must be on the boundary of P. By the same argument, g_1 and g_3 must be vertices of P. $\square$

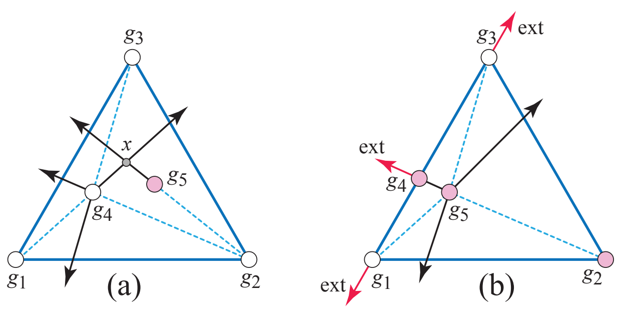


Figure 1: In this and following figures, guards are indicated by hollow circles. (a) Generic placements of g_4, g_5 produce a 2-dark point x. (b) The exceptional case, with dark rays exterior to P.

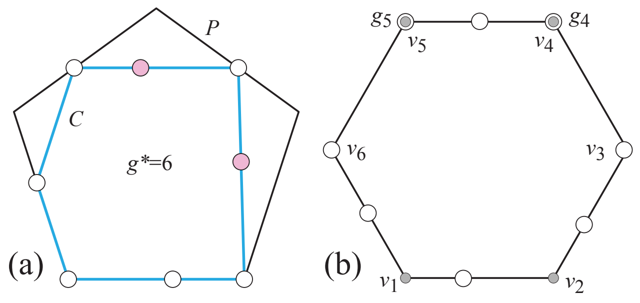


Figure 2: (a) The two pink guards are not included in $g^* = |G^*|$. (b) v_1, v_2 are darkened but have no guard; g_4, g_5 are both guards and darkened vertices. So $d = 4$ and $g^P = n + \frac{1}{2}d = 8$.

We now sketch the main idea of the $4n - 2$ upper bound. Consider a placement of guards in P such that there are no 2-dark points in P. Our goal is to prove that there are at most $4n - 2$ guards. Let C be the convex hull of the guards. We will show in Lemma 4 that the number of guards on ∂C, not counting collinear guards interior to P, is at most $2n$. Triangulating C leads to at most $2n - 2$ triangles. Lemma 3 then shows that there is at most one extra guard inside each triangle, which leads to the $4n - 2$ upper bound. To make this rigorous, we must take into account collinear guards and the exceptional case of Lemma 3.

We first shrink P so that it maximally touches C, as follows. Move each edge of P parallel to itself toward the interior until it hits a guard. If an edge e only has a guard at one endpoint, then rotate e about that endpoint toward the interior until it hits another guard. The reduced polygon contains all the guards, has no 2-dark point, and has at most n vertices, so it suffices to prove the bound on the number of guards for the reduced polygon. Henceforth we assume every edge of P has either one or more guards in its interior, or a guard at its endpoint (or at both endpoints).

The proof requires careful handling of collinear guards: a guard is called ***collinear*** if it lies on a line between two other guards.

Define $\boldsymbol{G^*}$ as the set of guards on ∂C, but excluding those guards that are collinear and not on ∂P. So collinear guards on ∂P are in G^*, but collinear guards on ∂C and internal to P are excluded from G^*. See Fig. 2. Equivalently, G^* consists of the guards on ∂P together with any guard that is a corner of C in the interior of P. Define $\boldsymbol{g^* = |G^*|}$. This is the key count that is needed to complete the upper-bound proof.

Lemma 4 *The number of guards g^* as defined above is at most $2n$.*

Proof. Let g^P be the number of guards on ∂P and let c be the number of guards that are corners of C in the interior of P. As noted above, $g^* = g^P + c$. We will bound g^P and c separately. Both bounds are in terms of the number of darkened vertices, where a vertex v of P is ***darkened*** if guards on ∂P generate a dark ray through v.

We first bound g^P. The constraint that limits g^P is that a vertex v cannot be darkened from both incident edges, as that would render v a 2-dark point.

The idea is to count guards and darkened vertices per edge. A guard internal to an edge counts towards the edge, and a vertex guard counts half towards each incident edge. More precisely, for an edge e, let $g(e)$ be the number of guards internal to e plus half the number of vertex guards on e. Then $g^P = \sum_e g(e)$.

Fig. 3 shows the possibilities: $g(e) = 2$, either from two internal guards, or one internal guard and two endpoint guards; $g(e) = 1\frac{1}{2}$ from one endpoint guard and one internal guard; or $g(e) = 1$ from one internal guard or two endpoint guards.

These are the only possibilities: (a) An edge cannot have four or more guards, as then the extreme points would be at least 2-dark. (b) And an edge can only have three guards when two are at the endpoints of the edge: an endpoint without a guard would be rendered 2-dark by the three guards on the edge. (c) An edge cannot have just a guard at one endpoint, because the shrinking procedure would rotate that edge about the endpoint until it hit another guard.

Next we observe from Fig. 3 a relationship between $g(e)$ and $d(e)$, the number of dark rays on edge e generated by guards on e: if $g(e) = 2$ then $d(e) = 2$; if $g(e) = 1\frac{1}{2}$ then $d(e) = 1$; and if $g(e) = 1$ then $d(e) = 0$. Equivalently, $d(e) = 2(g(e) - 1)$.

Finally, we note that d, the number of darkened vertices, is $\sum_e d(e)$, since each dark ray on e darkens an endpoint of e, and no vertex can be darkened from both incident edges.

Putting these together,

$$d = \sum_e d(e) = \sum_e 2(g(e)-1) = 2\sum_e g(e) - 2n = 2g^P - 2n$$

which gives $g^P = n + \frac{1}{2}d$. For example, for even n, placing a guard at every vertex and a guard in the interior of every other edge darkens every vertex, so $g^P = \frac{3}{2}n$.

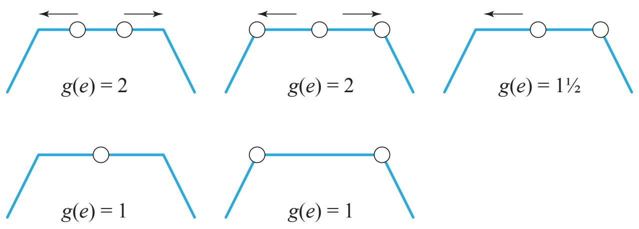


Figure 3: Edge counts. Arrows indicate darkened vertices.

We next bound c, the number of guards strictly internal to P that are corners of C. Let g_0 be such a corner guard. Moving left and right on C, let g_1 and g_2 be the first guards that are on ∂P, say on edges e_1 and e_2. Note that there cannot be another vertex of C internal to P between g_1 and g_2, as then two dark rays would cross inside P: see Fig. 4(a). Also note that g_0 is not collinear with g_1 and g_2, because we are counting g^*, which excludes collinear guards on C. Since every edge has a guard, edges e_1 and e_2 must be incident at a vertex v of P, and v has no guard (because otherwise g_0 would be internal to C). The dark rays incident to g_0 from g_1 and g_2 cross e_1 and e_2 as shown in Fig. 4(b). So v cannot be darkened by the guards on e_1 or e_2 otherwise again two dark rays would cross.

Thus each guard g_0 counted in c corresponds to a non-darkened vertex, so $c \le n - d$.

In total,

$$g^* = g^P + c \le n + \frac{1}{2}d + (n-d) = 2n - \frac{1}{2}d \le 2n\ .$$

Equality is achieved when there is one guard internal to each edge, and one guard inside P between each consecutive pair, and no collinear guards nor darkened vertices of P. See Fig. 4(c). □

Theorem 5 *The number of guards g that can be placed in a convex n-gon so that no two dark rays intersect inside is at most $g = 4n - 2$.*

Proof. Consider a placement of guards inside P that avoids 2-dark points. We use G^* and g^* as defined above. By Lemma 4, $g^* \le 2n$. Triangulate the guards in G^*. By definition of G^*, this includes collinear guards on ∂P but excludes collinear guards internal to P.

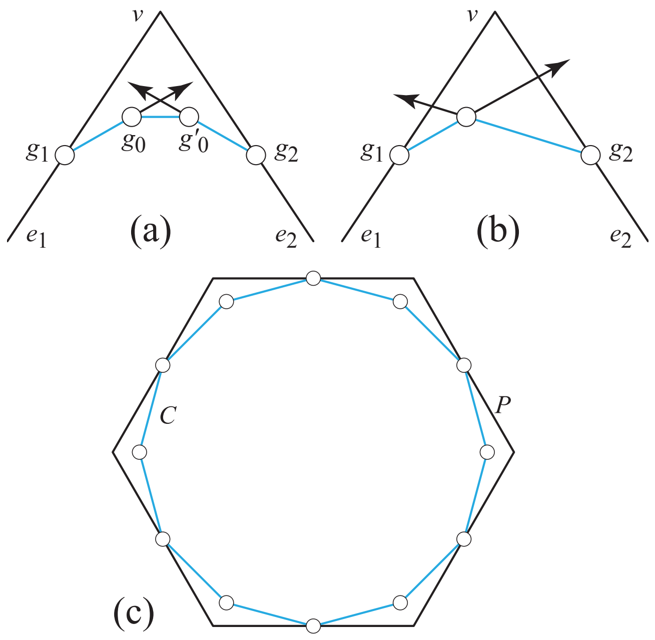


Figure 4: (a) g_0 and g_0' create intersecting dark rays in P. (b) v cannot be a darkened vertex. (c) The upper bound $g^* = 2n$ can be achieved.

There are at most $2n-2$ triangles in this triangulation. By Lemma 3, there is at most one extra guard in each triangle, for a total of at most $2n + (2n-2) = 4n-2$ guards, *so long as* we rule out the exceptional case of Lemma 3 where a triangle of guards can contain two extra guards. But that exception only happens when one of the extra guards is on ∂P, and all the guards on ∂P were already included in G^*. □

3 Lower Bound

The challenge is to locate $g = 4n - 2$ guards so that there are no 2-dark points in P, thus proving the lower bound of Theorem 2.

We first illustrate a placement in a triangle of $g = 10$ guards without 2-dark points, i.e., so that no two dark rays intersect inside the triangle. We then introduce the general strategy for the triangle, and hint at the strategy for convex n-gons, but proofs are deferred to Appendix A.3.

3.1 $g = 4n - 2$ guards achievable for triangle

Fig. 5 illustrates a placement of 10 guards in a triangle P such that all dark-ray intersections are strictly exterior to P. Although it is difficult to verify visually, even enlarged, a calculation described in the Appendix verifies that all dark-ray intersections lie strictly exterior to the triangle. This demonstrates $g = 4n-2$ is achievable for triangles.

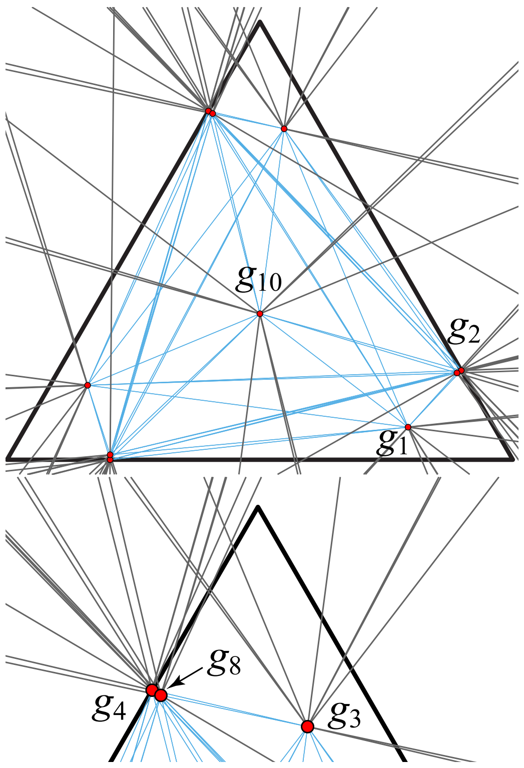


Figure 5: $g = 10$ guards 9-covering a triangle. Apex enlargement below. Indexing follows Fig. 6.

Several features of this construction will repeat for general n-gons:

(1) n guards are on edges of P.

(2) $2n$ guards are on the hull ∂C (the maximum by Lemma 4).

(3) Three guards are placed near each vertex,

(4) Two of the three guards near a vertex are nearly co-located.

(5) There is one extra guard in each triangle of a triangulation of P (this is g_{10} in Fig. 5).

This construction leads to 3 guards near each of P's n vertices, plus $n-2$ guards in the triangles of a triangulation, yielding $g = 4n-2$. Note that the triangulation is of the n-gon P, not the $2n$-gon convex hull C used in the proof of Theorem 5.

Idea of the construction in Fig. 5. Before turning to the general construction, we first provide intuition for the triangle construction, illustrated in Fig. 6. The triangle is partitioned into six sectors with g_{10} in the center. Three guards are placed in the yellow sectors near each vertex, so that the dark rays they generate at g_{10} exit through the empty white sectors. First, two of three guards are placed as illustrated: g_2, g_4, g_6 on triangle edges, and g_1, g_3, g_5 slightly inside the adjacent edges. The final three guards will be placed inside the convex hull of $g_1, \ldots, g_6$, but their locations are tightly constrained. The guards placed so far define three dark wedges apexed at guards g_1, g_3, g_5, where the wedge apexed at g_i contains all the dark rays at g_i. The last three guards g_7, g_8, g_9 are placed quite close to the even-index guards g_2, g_4, g_6 so that none of their dark rays enter the dark wedges. For further explanation, see Section A.3. The construction works for any triangle: there are no shape assumptions.

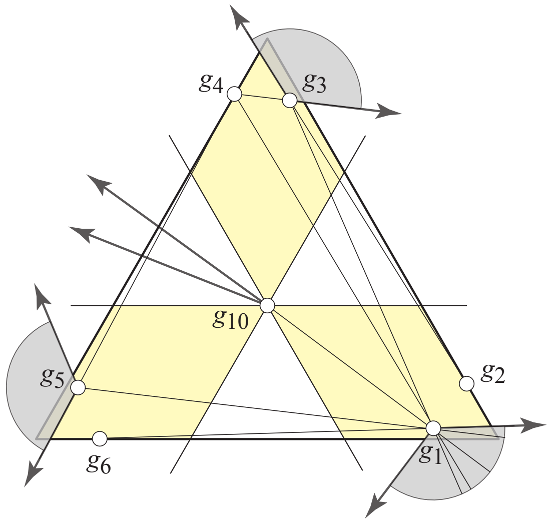


Figure 6: Dark rays from g_{10} exit through empty white sectors. Dark wedges apexed at g_1, g_3, g_5 contain the dark rays from all other guards, illustrated for the g_1 wedge.

The conclusion of the lower bound construction in the Appendix (Section A.3) is this theorem:

Theorem 6 *It is possible to place $4n-2$ guards in a convex n-gon P so that all dark-ray intersections lie strictly exterior to P.*

Theorems 5 and 6 establish the tight bounds in Theorem 2.

4 Simple Polygon

We mentioned in the Introduction that the variant we are exploring—multiple coverage and guards-blocking-guards—is not a natural fit for arbitrary simple polygons. In a convex polygon P, each pair of guards sees all of P except for their dark rays, whereas in an arbitrary polygon, guard visibility is also blocked by reflexivities of ∂P.

4.1 Necessity

The comb example that establishes necessity of $\lfloor n/3 \rfloor$ guards to cover a simple polygon of n vertices, also shows the necessity of $k\lfloor n/3 \rfloor$ guards to cover to depth k—since no guard can see into more than one spike of the comb, each of the $\lfloor n/3 \rfloor$ spikes needs at least k distinct guards. In fact, if the comb has at least two spikes, then $k\lfloor n/3 \rfloor$ guards also suffice. The general construction for $k \geq 2$ is illustrated in Fig. 7 for depth $k = 4$ and $n = 9$.

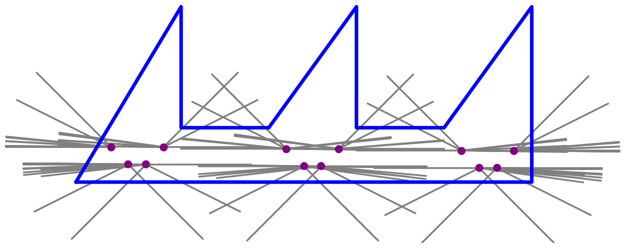


Figure 7: $4 \cdot 3 = 12$ guards suffice to 4-cover the comb of 9 vertices.

Place k guards in a convex arc below each spike of the comb so that none of the dark rays generated by these guards enters any spike. Points in a spike are covered to depth k by the k guards below it. Although many dark rays cross in the base corridor of the comb, slight vertical staggering of the convex arcs of k guards ensures that no corridor point is at the intersection of three dark rays, which ensures coverage to depth k for $k \geq 2$ and at least two spikes.

4.2 Sufficiency

For sufficiency, we have not obtained a tight bound: To cover a simple polygon P of n vertices to depth k, we show that $g = (k + 2)\lfloor n/3 \rfloor$ guards suffice. First triangulate P, 3-color, and choose the smallest color class, which has cardinality at most $\lfloor n/3 \rfloor$ [Fis78]. In Fig. 8, say we select color 1. If a color-1 vertex v is convex, then define a cone C apexed at v bounded by the edges incident to v. If a color-1 vertex v is reflex, then define C to be the “anticone” at v: the cone apexed at v and bound by the extensions of the incident edges into the interior.

To cover P to depth k, place $k + 2$ guards along a convex arc near a color-1 vertex v, and inside v's cone. In the figure, we aim to 3-cover and so place 5 guards in each cone. Now it is clear that the $k+2$ guards at color-1 vertex v see into all the triangles incident to v. These guards generate crossing dark rays, but by perturbing the locations of the guards we can avoid three dark rays meeting in P. The result is coverage to depth 2 less than the number of guards at each color-1 vertex:

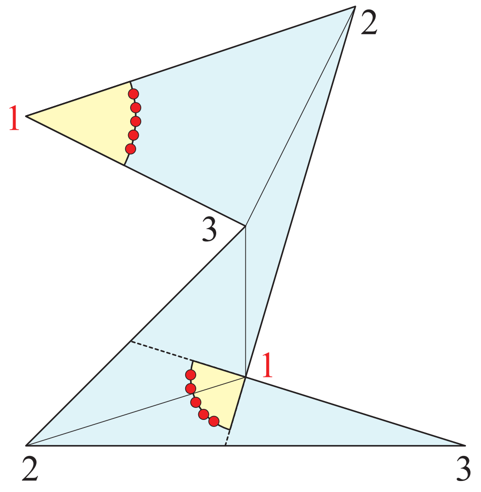


Figure 8: Cones at the color-1 reflex vertices each contain $k + 2$ guards. Here the 5 guards achieve a 3-cover.

Theorem 7 *To cover a simple polygon of n vertices to depth k, $g = k\lfloor n/3 \rfloor$ guards are sometimes necessary, and $g = (k + 2)\lfloor n/3 \rfloor$ guards always suffice.*

5 10 Guards in a Wedge

Finally, in Appendix A.5 we establish a tight bound for a wedge, which can be viewed as an unbounded 2-sided convex polygon with one vertex and two rays:

Lemma 8 *Covering a wedge to depth k requires the same number of guards as it does to cover a triangle to depth k, except that to 3-cover requires 4 guards. In particular, $g = 10$ guards can cover to depth 9.*

The surprising part of this result is that 10 guards can be placed in a wedge without creating 2-dark points—despite the fact that our triangle construction (see Fig. 6) fails for a wedge because it has 2-dark points just outside each triangle edge.

6 Open Problems

1. Investigate bounds or the complexity (NP-hard?) of placing points in a simple polygon so that no two dark rays intersect. (As noted in Section 4, the connection between this problem and k-guarding fails for non-convex polygons.)
2. Close the simple polygon gap in Theorem 7.
3. Can the tight bound for a wedge in Lemma 8 be generalized to tight bounds for unbounded convex polygons with two rays joined by a chain of $n - 1$ vertices and $n - 2$ edges?

Acknowledgements. We benefited from suggestions of three referees.

References

[AAM21] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The art gallery problem is ∃ℝ-complete. *J. ACM*, 69(1), 2021.

[BBC+94] Patrice Belleville, Prosenjit Bose, Jurek Czyzowicz, Jorge Urrutia, and Joseph Zaks. *K*-guarding polygons on the plane. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 381–386, 1994.

[BEK13] Daniel Busto, William S Evans, and David G Kirkpatrick. On *k*-guarding polygons. In *Proc. 25th Canad. Conf. Comput. Geom.*, 2013.

[DO11] Satyan Devadoss and Joseph O'Rourke. *Discrete and Computational Geometry.* Princeton University Press, 2011.

[Fis78] Stephen Fisk. A short proof of Chvátal's watchman theorem. *J. Combin. Theory Ser. B*, 24:374, 1978.

[O'R87] Joseph O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, NY, 1987. `http://cs.smith.edu/~jorourke/books/ArtGalleryTheorems/`.

[Sal09] Ihsan Salleh. *K*-vertex guarding simple polygons. *Comput. Geom. Theory Appl*, 42(4):352–361, 2009.

A Appendix

A.1 4-guarding a Triangle

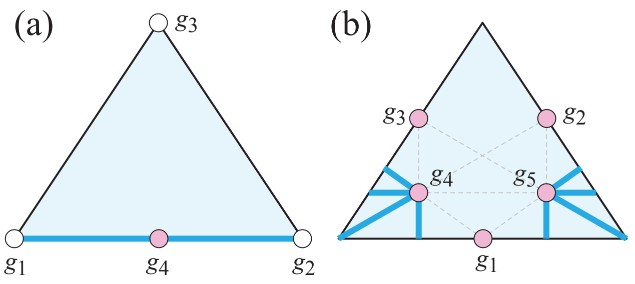


Figure 9: Five guards needed to 4-cover. (a) All strictly interior points are 4-covered, but the blue segments to either side of g_4 are only 3-covered. (b) Points on the dark rays (blue segments) incident to g_4 and g_5 are 4-covered; all other points are 5-covered.

A.2 Regime (2) Lemma

Lemma 9 *Any placement of $n+1$ guards in a convex n-gon P results in a dark point in P.*

Proof. If a guard g_0 is strictly internal to P, then there is a dark ray at g_0 generated by every other guard. So it must be that all guards are on ∂P.

View each edge of P as half-open, including its clockwise endpoint but not its counterclockwise endpoint. So the edges are disjoint and their union is ∂P. Every edge e can contain at most one guard: If e contains two or more, one, g_1, is interior to e and so there is a dark ray at g_1 along e. So there can be at most n guards while avoiding dark points. □

A.3 General Lower Bound Construction

Example: Square. Before commencing with the general construction, we illustrate it with a square. Placing $4n-2=14$ guards in a square without any 2-dark points follows the same construction as with the triangle in Fig. 5: 3 guards near each vertex, and $n-2=2$ "elbow" guards ℓ_i determined by a special triangulation, in this case just a diagonal of the square. See Fig. 10. Coordinates may be found in the Appendix (Section A.6).

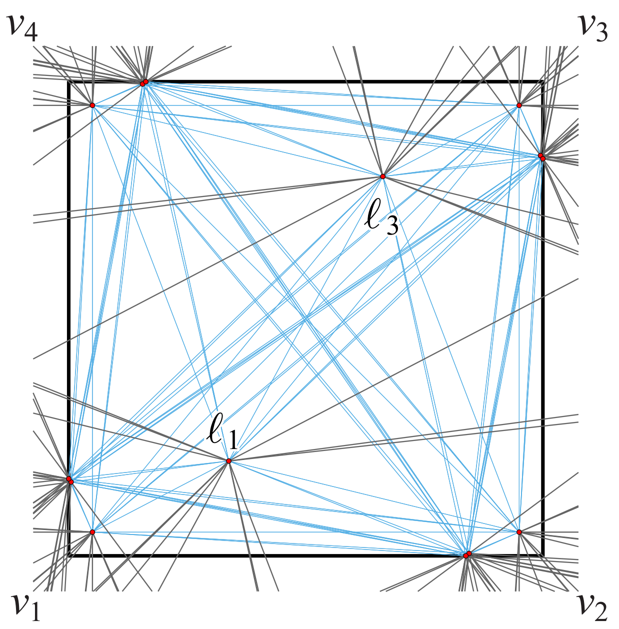


Figure 10: 14 guards covering to depth 13. Triangulation diagonal is v_1v_3. Elbow guards ℓ_1, ℓ_3. Vertex guards x_i, y_i, z_i near the four corners.

Overall Construction. The overall plan of the construction is the same as for a triangle and a square:

$3n$ guards, 3 near each vertex, plus one guard per triangle in a triangulation of P of $n-2$ triangles. The three guards to be placed near v_i will be called ***vertex guards***. The triangulation is a ***serpentine*** triangulation formed by a ***zigzag path*** that visits all the vertices, as illustrated in Fig. 11. The single guard in each triangle will be called an ***elbow*** guard.

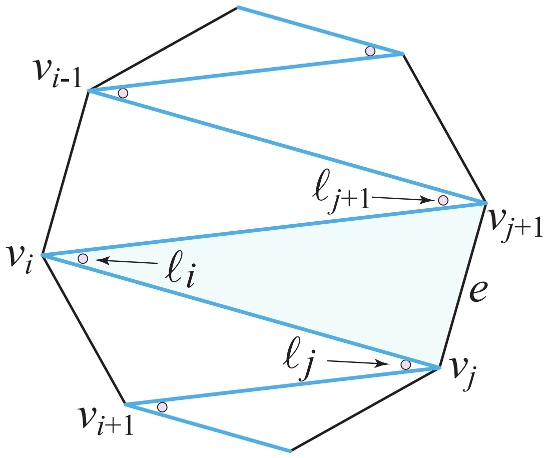


Figure 11: Zigzag triangulation and elbow guards ℓ_i.

Notation. We label the vertices in counterclockwise (ccw) order: $v_0, \ldots, v_{n-1}$ with index arithmetic modulo n. Thus "before" means clockwise (cw) and "after" means ccw. Let v_i be one of the $n-2$ internal vertices of the zigzag path. Then v_i is the apex of a triangle T_i bounded by two edges of the zigzag path plus a ***base*** that is an edge of the polygon. The elbow guard of T_i, which we denote ℓ_i, will be placed close to vertex v_i. For ease of notation, we will focus on one triangle with apex v_i and base v_jv_{j+1}. In each edge of P we place two "dividing points" that are used to separate wedges of dark rays. The dividing points adjacent to v_i are labeled m_i (on the minus (cw) side) and p_i (on the plus (ccw) side). See Fig. 12.

Note that there are two vertices of P with no elbow guard, and consequently either ℓ_j or ℓ_{j+1} (or both) might not exist. For example, in Fig. 10, neither ℓ_2 nor ℓ_4 exist.

Dark-ray Wedges. The elbow guard ℓ_i will be located close to v_i, and v_i's three vertex guards even closer to v_i. We first place the elbow guards and define "safe regions" for vertex guards so that the dark rays incident to elbow guards lie in disjoint "dark ray wedges." Exact placement of vertex guards will be described later.

Let e be the base edge of T_i, $e = v_jv_{j+1}$. Then the three portions of e demarcated by p_j, m_{j+1} each are crossed by wedges of dark rays incident to elbow guards. The central portion of e is crossed by rays generated by v_i's vertex guards through ℓ_i (blue). The v_jp_j segment of e is crossed by the rays at ℓ_j, generated by all the vertex guards and elbow guards associated with vertices ccw from v_{i+1} to v_{j-1}, and symmetrically the $m_{j+1}v_{j+1}$ segment of e is crossed by dark rays at ℓ_{j+1}, generated by all the vertex guards and elbow guards associated with vertices ccw from v_{j+2} to v_{i-1}.

From the viewpoint of ℓ_i, there are three dark wedges emanating from it, one crossing p_jm_{j+1} and two (shown in pink) crossing v_im_i and v_ip_i, before and after v_i.

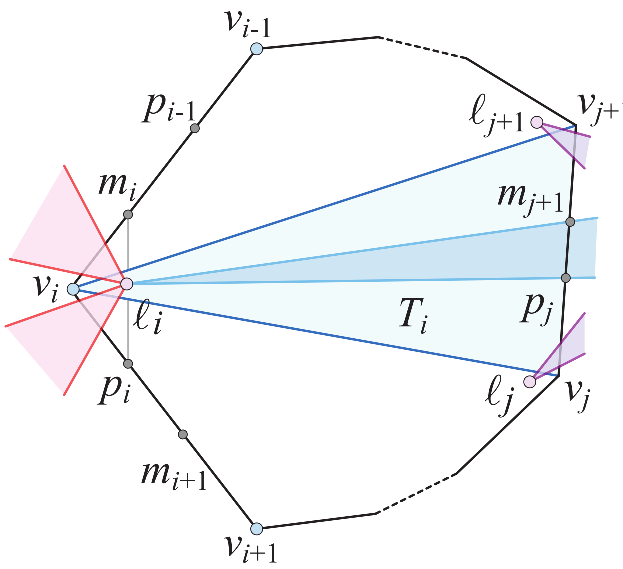


Figure 12: The dark-ray wedges that cross $e = v_jv_{j+1}$ and the dark-ray wedges emanating from ℓ_i.

Locating ℓ_i. We now describe how to place each ℓ_i so that the dark-ray wedges illustrated in Fig. 12 indeed contain the claimed rays, and create a "safe region" for v_i's vertex guards.

Place ℓ_i at the intersection of two lines: the line m_ip_i, and the line through v_i and the midpoint of p_jm_{j+1}.

Let b_i be the point where the line through p_j and ℓ_i exits P. Observe that b_i lies in the segment v_im_i. Our mnemonic is that b_i is just "before" v_i. Let a_i be the point where the line through m_{j+1} and ℓ_i exits P. Then a_i lies in the segment v_ip_i, just after v_i.

For a vertex v_i that has an elbow guard, define its ***safe region*** R_i to be the convex quadrilateral $b_iv_ia_i\ell_i$, which is contained in the triangle $m_iv_ip_i$. For a vertex v_i without an elbow guard (the first and last vertices of the zigzag path), its safe region is the triangle $m_iv_ip_i$. Observe that the safe regions are pairwise disjoint.

Claim 1 *If vertex guards for v_i are placed in R_i then the dark rays incident with elbow guards lie in the wedges as specified above and do not enter the safe regions.*

Proof. Consider the dark rays incident to ℓ_i. Since v_i's vertex guards lie in the wedge $a_i\ell_ib_i$, they generate dark rays at ℓ_i that lie in the complementary wedge $m_{j+1}\ell_ip_j$. Vertex guards and elbow guards associated with vertices ccw from v_{i+1} to v_j lie in the wedge $p_i\ell_ip_j$ so they generate dark rays at ℓ_i that lie in the complementary wedge $m_i\ell_ib_i$ (yellow wedges in Fig. 13). Similarly vertex and elbow guards associated with vertices ccw from v_{j+1} to v_{i-1} lie in the wedge $m_{j+1}\ell_im_i$ so they generate dark rays at ℓ_i that lie in the complementary wedge $a_i\ell_ip_i$. (green wedges in Fig. 13). □

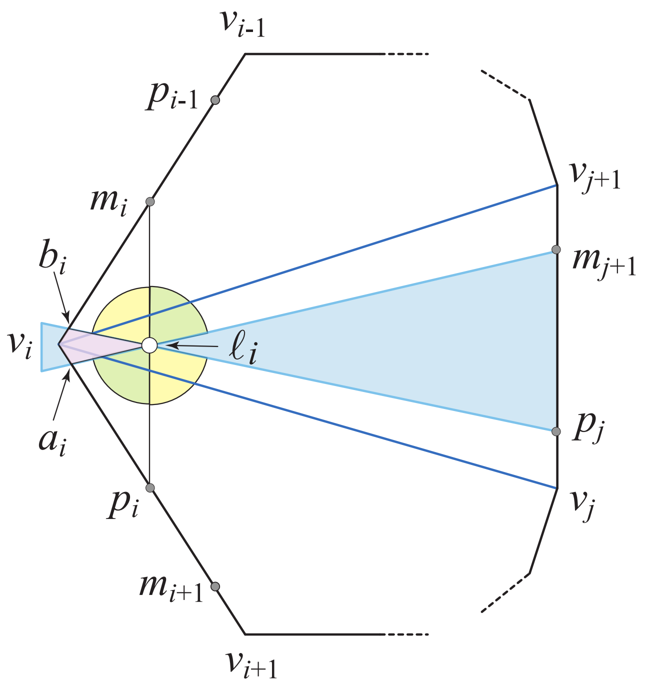


Figure 13: Constraints on locating ℓ_i, and for locating vertex guards in a safe region $R_i = b_iv_ia_i\ell_i$.

Locating 3 vertex guards. Call the three v_i vertex guards x_i, y_i, z_i. We will place them in that order, inside the safe region R_i. x_i will be placed on an edge of P, and x_i and y_i will be on the convex hull C of the guards, with z_i strictly inside C.

The following construction references a_i and b_i so it applies to the case when ℓ_i exists. But for a vertex v_i without an elbow guard, the same construction works with m_i and p_i in place of b_i and a_i.

Construct a triangle with apex v_i and two points on ∂P strictly inside the safe region R_i. Place x_i at the corner of this triangle on edge v_iv_{i-1}, and place y_i on the base of the triangle and on the p_i side of the line $v_i\ell_i$. Observe that all the elbow guards are inside the resulting hull C. Because x_i is the only guard on its edge, there are no dark rays incident to x_i inside P.

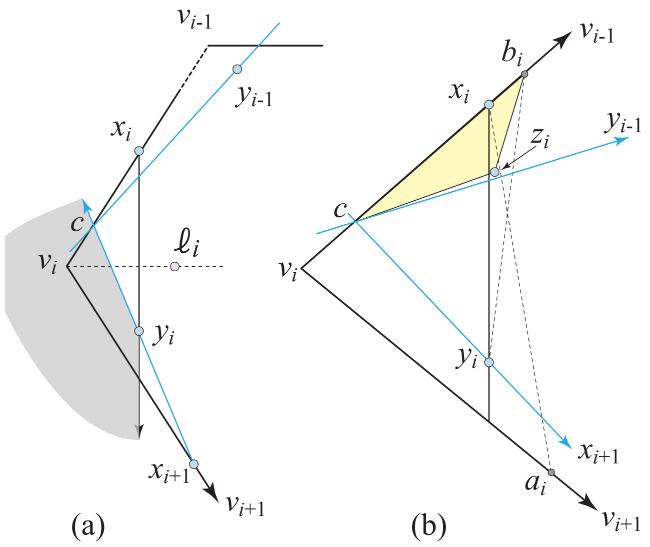


Figure 14: (a) Locating x_i and y_i. Wedge of dark rays apexed at y_i shaded. (b) Locating z_i so that dark rays incident to z_i exit P safely.

Because y_i lies on C with neighbours x_i and x_{i+1}, all the dark rays incident to y_i lie in the complementary wedge bounded by the lines y_ix_i and y_ix_{i+1}, and including v_i (gray in Fig. 14(a)). Note that no other dark rays intersect this wedge because it lies inside the safe region.

We now place z_i. Let c be the point where the line $x_{i+1}y_i$ intersects the edge v_iv_{i-1}. See Fig. 14(a).

We will ensure that the dark rays incident to z_i—except for the one generated by x_i—lie in the wedge cz_ib_i (yellow in Fig. 14(b)). This implies that these rays do not intersect any other dark rays.

We place z_i:

1. inside C,
2. on the x_i side of lines y_ib_i and $y_{i-1}c$,
3. on the y_i side of line x_ia_i.

Observe that these constraints determine a non-empty region for z_i.

Conditions 1 and 3 ensure that the dark ray incident to z_i generated by x_i hits the edge v_iv_{i+1} in the segment between y_i's dark wedge and a_i, so it intersects no other dark ray.

Conditions 1 and 2 ensure that, if we ignore x_i, then z_i lies on the convex hull C' of the guards, with neighbours y_i and y_{i-1}. Therefore the dark rays incident to z_i lie in the complementary wedge—apexed at z_i and exterior to C'—which lies inside the wedge b_iz_ic, as required.

We note that, although our construction places guards quite close together, the coordinates have polynomially-bounded bit complexity, since we used a finite sequence of linear constraints. By contrast, irrational coordinates may be required for the conventional art gallery problem in a simple polygon [AAM21].

Note that at no point do we rely on the metrical properties of P, so the construction works for all convex polygons:

Theorem 6 *It is possible to place $4n-2$ guards in a convex n-gon P so that all dark-ray intersections lie strictly exterior to P.*

To repeat our earlier claim, Theorems 5 and 6 establish the tight bounds in Theorem 1.

A.4 General Position Guards

At several junctures we claimed we can avoid 3-dark points inside P by perturbing the guard locations to be in "general position." Although this follows from general perturbation results, we give a straightforward inductive construction.

We show how to place g guards in a specified open region of the plane (a convex polygon in regime (3), or near the vertex of a vertex cone in the situation of Section 4) while avoiding 3-dark points anywhere in the plane.

Place the guards sequentially. After placing i guards, let $\mathcal{A}_i$ be the arrangement of lines determined by: (a) pairs of guard points; and (b) a guard point and a 2-dark point at the intersection of two dark rays. (For $i \leq 3$ noncollinear guards, there are no 2-dark points.) Place the $(i+1)$-st guard at any point in the open region not on a line of $\mathcal{A}_i$. This is possible since the region is open. Note that this avoids three collinear guards and also avoids three dark rays crossing. Now update the arrangement to $\mathcal{A}_{i+1}$ and repeat.

A.5 10 Guards in a Wedge

Define a ***wedge*** as the region of the plane bounded by two rays from a convex vertex a, i.e., a cone with apex a. The connection between k-guarding and dark points (Observation 1) still holds, and the main issue is the analogue of Theorem 2—what is the maximum number of guards that can be placed in a wedge without creating 2-dark points? For a triangle, the bound is $4n-2=10$. In this section we prove that the same bound holds for a wedge.

The upper bound of 10 is easy: If we could place 11 guards in a wedge without 2-dark points, then we could simply cut off the empty part of the wedge to create a triangle with 11 guards and no 2-dark points, a contradiction to the Theorem 5 upperbound.

However, the lower bound of 10, i.e., a placement of 10 guards without 2-dark points, does not carry over from our triangle construction, because there were dark ray intersections beyond every edge of the triangle. Nevertheless, we now show this bound is tight, with the example illustrated in Figs. 15 and 16. We number the guards from bottom to top. Here is a description of the construction:

- g_1 is directly below the apex a, and far below.
- g_2 is slightly left of g_1, so that the upward dark ray at g_2 exits the wedge at a particular "safe" spot between g_7 and g_{10}.
- Guard pairs g_3, g_4, g_5, g_6, g_7, g_8 are symmetrically placed with respect to a vertical line L through a.
- Guards g_7, g_8 are located on the two edges of the wedge.
- g_{10} is on L near a, while g_9 is right of L.
- There are six guards on the convex hull C of the guards: $\{g_1, g_3, g_7, g_{10}, g_8, g_4\}$.
- g_5, g_6 are just slightly inside C.

We provide coordinates for the guards in Appendix A.6, and have verified that there are no 2-dark points in the wedge.

Note that this construction provides an alternative arrangement of guards for a triangle: Introduce a triangle edge bc below g_1, and apply an affine transformation to $\triangle abc$ to match Fig. 15.

We summarize the implications for k-guarding a wedge in this lemma.

Lemma 8 *Covering a wedge to depth k requires the same number of guards as it does to cover a triangle to depth k, except that to 3-cover requires 4 guards. In particular, $g = 10$ guards can cover to depth 9.*

Proof. If $k \leq 2$, a guard at the one vertex, or one guard on the interior of each edge, suffices. However, any placement of 3 guards creates a dark point in the wedge, so for $k \geq 3$, at least $k+1$ guards are needed to k-guard. For $k \leq 9$, the configuration just described shows that $k+1$ guards suffice—this covers the middle regime. For $k \geq 10$, $g = k+2$ guards are needed and suffice, from Observation (3) in Section 1.1 and its explanation in Section A.4. □

A.6 Guard Coordinates

We include here explicit coordinates for guards in a triangle, a square, and a wedge. In all cases, Mathematica code has verified that dark-ray intersections are strictly exterior.

Coordinates for 10 guards in an equilateral triangle, Fig. 5. Triangle corners are $(0, 200), (\pm 100\sqrt{3}, -100)$. Guard locations for the other g_i are symmetrical placements following Fig. 6.

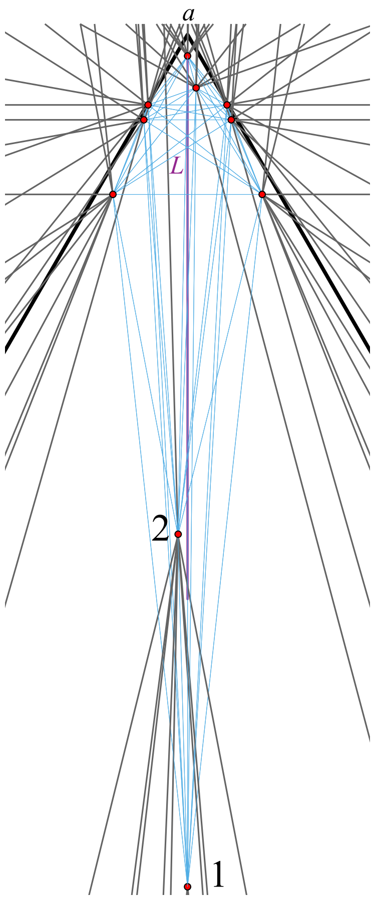


Figure 15: Wedge apex a, 10 guards with no 2-dark points.

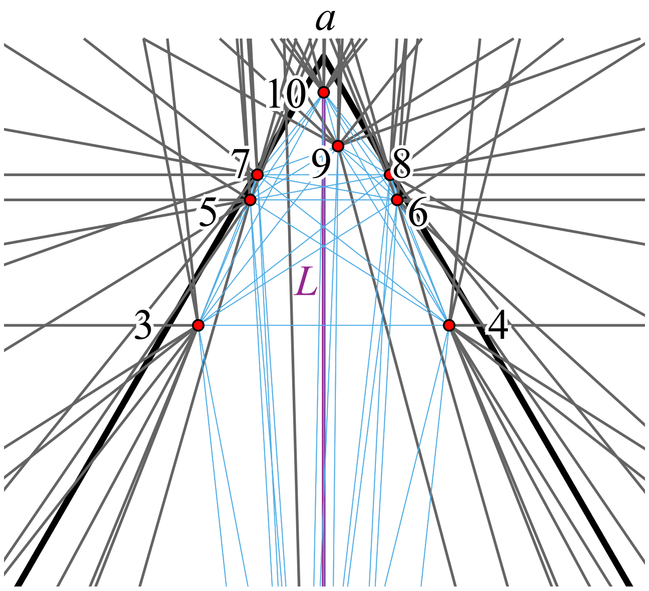


Figure 16: Closeup of upper portion of Fig.15.

g_i	$x,$	y
5	$-102.57,$	-96
6	$-102.6,$	-100
7	$-118,$	-49
10	$0,$	0

Coordinates for 14 guards in a square, Fig. 10. Square corner coordinates $(\pm 200, \pm 200)$. Guard locations $g_6, \ldots, g_{14}$ are symmetrical placements of g_3, g_4, g_5.

g_i	$x,$	y
1	$-65,$	-120
2	$65,$	120
3	$-180,$	-180
4	$-198,$	-137.7
5	$-200,$	-135

Coordinates for 10 guards in a wedge, Figs. 15 and 16. Apex at $(0, 200)$, apex angle $\pi/3$. Guard locations g_4, g_6, g_8 are symmetrical placements of g_3, g_5, g_7.

g_i	$x,$	y
1	$0,$	-600
2	$-9,$	-270
3	$-70,$	50
4	$70,$	50
5	$-41,$	120
6	$41,$	120
7	$-38.1,$	134
8	$38.1,$	134
9	$8,$	150
10	$0,$	180

Conflict-Free Chromatic Guarding of Orthogonal Polygons with Sliding Cameras

Yeganeh Bahoo[*] Onur Çağırıcı[*] Kody Manastyrski[*] Rahnuma Islam Nishat[†] Christopher Kolios[*] Roni Sherman[*]

Abstract

In *conflict-free chromatic guarding of a polygon*, every guard is assigned a color such that every point in the polygon, including the points on the boundaries, must see at least one unique color. The goal of this problem is to minimize the number of colors needed. In this paper, we study the conflict-free chromatic guarding of *simple orthogonal* polygons with *sliding cameras*, where cameras are allowed to slide along the length of the corresponding edge. We investigate two versions of the Conflict-free Sliding Camera problem: for orthogonal polygons without holes (CFSC), and for orthogonal polygons with holes (CFSC-H), we show that two colors are always sufficient and sometimes necessary for a CFSC, and give an $O(n \log n)$ time algorithm to compute a CFSC using two colors, where n is the number of vertices of the polygon. We give an $O(n \log n)$ time algorithm to obtain a CFSC-H using three colors. We also show that for a special case of CFSC-H two colors suffice.

1 Introduction

The polygon guarding problem is a well-studied problem in the field of computational geometry, which is also known as the *art gallery problem* [7, 18]. Given a polygon P, the goal of the polygon guarding problem is to find the minimum number of guards needed so that any point in P is visible to at least one guard. Two points p and q are *visible* to each other when the line segment pq, also known as *line-of-sight visibility*, does not intersect any edges of P. This problem has been studied in many different settings, such as for general polygons [10, 11, 17], for weak-visibility polygons [2, 3], and for orthogonal polygons [15, 16].

The very first version of this problem, introduced by Victor Klee [18] considered line-of-sight visibility. Later, variations of the problem were studied while assuming different visibility models, such as α-visibility [14], and π-visibility [19].

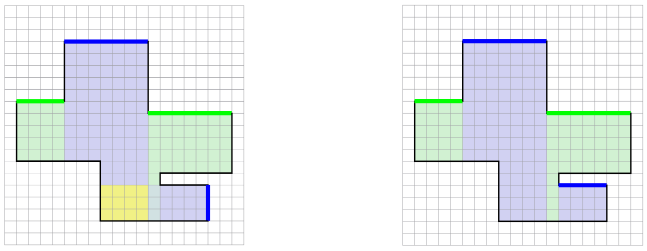

(a) (b)

Figure 1: (a) An assignment with conflicts: the yellow region is covered by two blue guards and no green guards; the blue-green region has a blue and a green guard and causes no conflict. (b) A conflict-free assignment.

Although Fisk applied graph coloring in the proof of bounds in art gallery problems [13], chromatic guarding of polygons is a more recent topic. In chromatic guarding problem, we look for a set of guards such that each point of the polygon is visible to a subset of the guards and assign a color to each guard from a set of available colors. The set of guards along with the color assignment is conflict-free if for each point of the polygon, there is at least one guard with a unique color [1]. Conflict-free chromatic guarding has applications in the assignment of radio frequencies to sensors placed on the vertices of the polygon to guide mobile robots in triangulating their positions in the polygon [1, 5].

Conflict-free chromatic guarding has been studied in the context of orthogonal polygons, and bounds have been given on *chromatic numbers* χ_P of an orthogonal polygon P, i.e., the minimum number of colors required to guard P without conflict. Given a polygon P, Bärtschi and Suri [1] showed that $\chi_P \in O(\log n)$, where n refers to the number of vertices of the polygon, by subdividing P into "weak-visibility suppolygons". Erickson and LaValle [12] showed that for orthogonal staircase polygons, the bound is $\chi_P \leq 3$.

In this work, we use *sliding cameras* as guards [4, 8, 9, 16], where a guard or camera is *directional* (i.e. it has directional view oriented towards the interior of the given polygon) and can travel along a boundary edge

[*]Department of Computer Sciece, Toronto Metropolitan University, Toronto, ON, Canada, {bahoo, cagirici, kody.a.manastyrski, ckolios, roni.sherman}@torontomu.ca

[†]Department of Computer Science, Mathematics, Physics and Statistics, University of British Columbia, Kelowna, BC, Canada, rahnuma.nishat@ubc.ca

of the polygon. See Figure 1(b) for an example, where sliding cameras or guards are assigned (only to horizontal) edges of the polygon. Note that two consecutive horizontal edges on the polygon have not been assigned cameras of the same color as that would create conflict at the boundary of the two guarded regions. Throughout the paper, the terms *sliding camera* and *guard* are used interchangeably.

Our contributions. We give upper and lower bounds on χ_P for both CFSC and CFSC-H. In Section 3, we prove that two colors are sometimes necessary and always sufficient for a conflict-free chromatic guarding of an orthogonal polygon without holes. Our bound on the chromatic number is tight. We also give an $O(n \log n)$ time algorithm that solves CSFO with two colors, where n refers to the number of vertices of P. In Section 4.1, we propose an $O(n \log n)$ time algorithm to solve a CSFO-H, using three colors ($\chi_P \leq 3$). Finally, in Section 4.2, we study a special case of CFSC-H, where each hole has a rectangular boundary and the order of the holes inside the polygon is *X-monotone*. In this case, we show that two colors are sufficient ($\chi_P = 2$). In all our algorithms, the outer boundary of the polygon and the boundaries of the holes are axis-parallel.

2 Preliminaries

We define the terminology used throughout the paper.

A simple *polygon* P is an enclosed area in the Euclidean plane bounded by a finite number of straight line segments that form a polygonal chain; each such straight line segment is called an *edge* of P, and a pair of edges meet in a *vertex*. In this paper, first we consider simple polygons; i.e. no pair of edges intersect except at their common endpoints (vertices). Starting from Section 4, we suppose that a polygon with holes is given. A polygon with holes is a polygon enclosing several other polygons; the inner polygons are known as holes.

The *boundary* of P is the closed polygonal chain formed by the edges of P. We assume that the boundary is directed clockwise, i.e., while walking along the boundary of the polygon the interior would always be on the right. The vertices of a polygon are denoted by $v_1, v_2, \ldots, v_n$ in clockwise order. The edge that connects the pair of vertices v_i and v_{i+1} is denoted by e_i, where $1 \leq i \leq n$, and the edge from v_n to v_1 is denoted by e_n.

The polygon P is called an *orthogonal polygon*, if every pair of consecutive edges are perpendicular to each other. Throughout this manuscript, we assume that a given orthogonal polygon is oriented in a way such that each edge is parallel to either x-axis (horizontal), or y-axis (vertical). We classify the edges as north-facing, south-facing, east-facing and west-facing depending on the orientation of the perpendicular ray towards the interior of the polygon.

An *X-monotone* polygon is an orthogonal polygon that intersects any vertical line ℓ at most twice, where an intersection is either a point on a horizontal edge of the polygon, or an entire vertical edge.

Definition 1 *A* maximal monotonous south-facing chain *is a maximal set of south-facing edges $e_1, \ldots, e_k$ such that the x-coordinates of the starting points of the edges are in increasing order, and e_i and e_{i+1}, for each $i < k$, are connected by a vertical edge.*

Let $\mathbb{S}$ be the set of all maximal monotonous south-facing chains $S_1, S_2, \ldots, S_q$ in P. For any chain $S_i \in \mathbb{S}$, we denote the edges of S_i by $e^i_1, e^i_2, \ldots, e^i_{k_i}$ from left to right. The *visibility region* of a chain S of $\mathbb{S}$ is the region of P, including the boundary of P, that is visible to the guards assigned to the edges of S. We observe the following property of the set $\mathbb{S}$.

Lemma 1 *A sliding camera at each edge of $\mathbb{S}$ collectively guards the entire polygon P.*

Proof. From any point $p \in P$, if we draw a vertical line upward, the first edge e that the line intersects must be south-facing; hence, e must belong to a chain in $\mathbb{S}$. □

Lemma 1 forms the basis of our algorithms in this paper, where we find efficient ways to put a camera on each of the edges of $\mathbb{S}$ such that the number of colors needed for a conflict-free guarding of P is minimized.

We use blue, red and green for the three colors assigned to guards or cameras. When a guard is assigned a color, say red, we write *red guard* or *red camera.*

3 Orthogonal polygons without holes

In this section, we discuss the CFSC problem (for orthogonal polygons without holes) and give tight upper and lower bounds on the conflict-free chromatic number χ_P for them. We show that two colors are sometimes necessary and always sufficient to guard an orthogonal polygon with sliding cameras, thus giving a lower and an upper bound on the number of colors required to guard any orthogonal polygons. The lower bound holds for polygons with holes as well.

We first prove the lower bound on χ_P.

Theorem 2 *There exists an orthogonal polygon that requires sliding cameras of at least two colors for CFSC.*

Proof. Let P be the orthogonal polygon in Figure 2. By exhaustive search, we conclude that a sliding camera on any of the 8 edges of P cannot guard the whole polygon. If there exists only one color of gaurds, any combination of guards in this figure will result in a conflict as no consecutive edges of P can be assigned the same color. Therefore, we need at least two cameras. The visibility regions of any two cameras guarding P will intersect at least at the boundaries. Therefore, we must need at least two different colors for the cameras. □

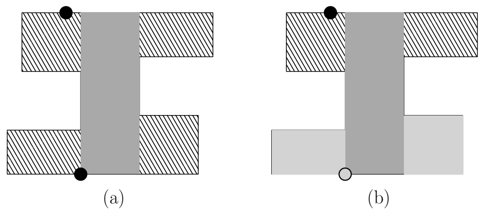


Figure 2: (a) Two black guards guarding the whole polygon; the patterned regions are visible to a single guard, and the solid region to both guards. (b) A conflict-free guarding by a black (patterned regions) and a gray guards. The dark solid region is covered by both guards.

We now show that two colors are always sufficient for a CFSC of any given orthogonal polygon. First, we describe an algorithm to obtain a CFSC for an X-monotone polygon, and then we generalize the idea to non-monotone polygons.

By definition, an X-monotone polygon P has exactly one maximal monotonous south-facing chain S. We place a blue camera on all the edges e_i of S, where i is odd, and a red camera on the edges e_i, where i is even; see Figure 3 for an example. We name this

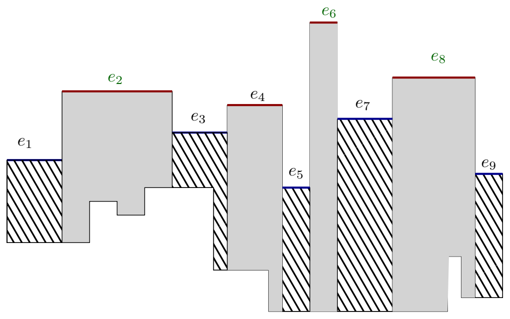


Figure 3: Two colors are sufficient to guard a monotone orthogonal polygon.

algorithm CFSC-MONOTONE. The following theorem proves the correctness and running time of the above algorithm. The proof is in the appendix.

Theorem 3 *Algorithm* CFSC-MONOTONE *computes a CFSC of an X-monotone polygon P without holes in $O(n)$ time using only two colors, which is optimal.*

Now consider the case where the input polygon P is not X-monotone. Then $\mathbb{S}$ has more than one chain. We then need some special data structure to assign colors to the edges in $\mathbb{S}$ efficiently.

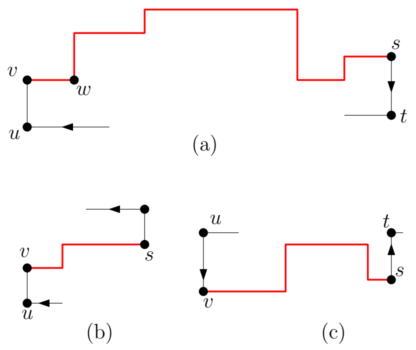


Figure 4: Maximal monotonous south-facing chain with (a) no upward connection, (b) one upward connection, and (c) two upward connections.

Before describing the data structures for our algorithm, we need to define some terminology. Let the vertical edges preceding and following a chain $S_i \in \mathbb{S}$ be (u, v) and (s, t), respectively, where v and s are the endpoints of the first edge e_1^i and last edge e_k^i of S_i, respectively. If u has a *higher* y-coordinate than v then we say that e_1^i is a *upward connection* S_i. Similarly, e_k^i is an upward connection of S_i if t has a higher y-coordinate than s. Clearly, S_i can have upward connections 0, 1, or 2 as shown in Figure 4.

We now describe the data structures we need. In the preprocessing step, we populate a list called $\mathcal{L}$ that stores pointers for each $S \in \mathbb{S}$ of the chains that are influenced by or influence the coloring of S. For this we build a trapezoidal map $\mathcal{T}$ and the associated search structure $\mathcal{D}$ with the south-facing edges of P using the algorithm proposed in [7]. Therefore, with each edge, the chain associated with $\mathbb{S}$ is given. Now for each *upward connection* v of each $S \in \mathbb{S}$, we traverse the search structure $\mathcal{D}$ to find the edge e and the associated chain S_i just above v. Let v be incident to edge e' in S. We then put (e', e, S_i) in the $\mathcal{L}$ entry of S, and put (e, e', S) in the corresponding entry for S_i. Building $\mathcal{T}$ and $\mathcal{D}$ takes expected $O(n \log n)$ time using the incremental randomized algorithm in [7], and finding the edge above each endpoint takes $O(\log n)$ time. Therefore, $\mathcal{L}$ can be populated from $\mathcal{D}$ in $O(n \log n)$ time.

Figure 5 shows an orthogonal polygon without holes, and Table 1 shows the corresponding data structure $\mathcal{L}$.

We now briefly describe the algorithm for assigning two-color sliding guards to the edges of $\mathbb{S}$ to obtain a conflict-free guarding of P. We call the algorithm CFSC-TWOCOLORS.

We pick any chain S from $\mathbb{S}$ and assign guards to the chain by putting a red camera on the edges e_i, where i is odd, and a blue camera on the edges e_i, where i is even. For each entry (e, e', S') in the influence list $\mathcal{L}$ of S, where e is an edge of S and $c(e)$ denotes the color

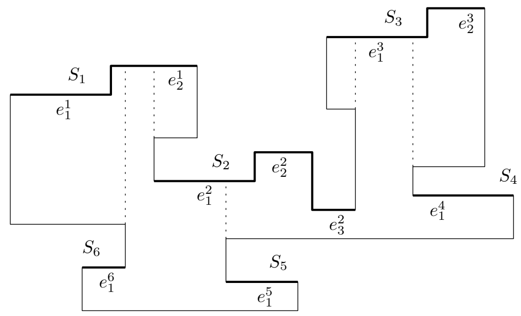


Figure 5: The chains in $\mathbb{S}$ are shown, with the edges, and the upward connections with dotted lines.

S_i	influence list
S_1	(e_1^6, e_2^1, S_6), (e_1^2, e_2^1, S_2)
S_2	(e_1^2, e_2^1, S_1), (e_1^5, e_1^2, S_5), (e_1^3, e_3^2, S_3)
S_3	(e_3^2, e_1^3, S_2), (e_1^3, e_1^4, S_4)
S_4	(e_1^4, e_1^3, S_3)
S_5	(e_1^5, e_1^2, S_2)
S_6	(e_1^6, e_2^1, S_1)

Table 1: Data structure $\mathcal{L}$ for the polygon in Figure 5.

assigned to edge e, we put $(c(e), e', S')$ in a queue Q. In the next step, when we remove that entry $(c(e), e', S')$ from Q, we assign the opposite color of $c(e)$ to e' of S', and continue to assign alternating colors to the edges on each side of S'. After that, we put all the chains in the influence list of S' that have not been colored into Q. We stop when the queue is empty.

Figure 6 shows how the algorithm works by showing the first step. The following theorem proves the correctness and running time of the algorithm. See the appendix for the complete proof.

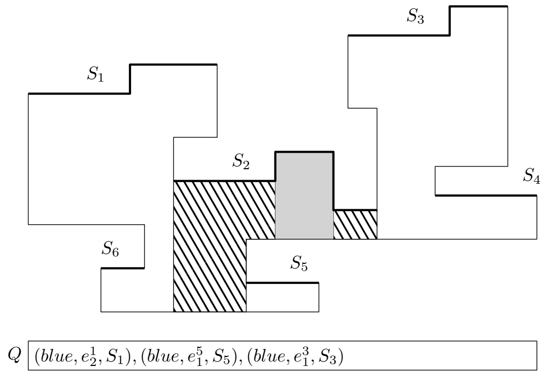

S_3
S_1
S_2
S_4
S_6
S_5
Q
$(blue, e_2^1, S_1), (blue, e_1^5, S_5), (blue, e_1^3, S_3)$

Figure 6: A CFSC with two colors, after assigning colors to the edges of $S_2 \in \mathbb{S}$; and the queue Q after coloring S_2.

Theorem 4 *Algorithm* CFSC-TwoColors *computes a CFSC of an orthogonal polygon P without holes in $O(n \log n)$ time using two colors.*

Proof Sketch. After coloring a chain S, if we remove the visibility region of S from P, we get disjoint sub-polygons for each of which only one endpoint of one south-facing edge has been assigned a color. Since the corresponding chain S' is placed in Q before any other chains in that sub-polygon, S' also gets a conflict-free coloring. By inductively applying the above logic, we can prove the correctness. Assigning colors to all the south-facing edges takes $O(n)$ time. Building $\mathcal{T}$, $\mathcal{D}$, and then populating $\mathcal{L}$ from $\mathcal{D}$ takes $O(n \log n)$ time. Therefore, the total time is $O(n \log n)$.

4 Orthogonal polygons with holes

In this section, we study the problem *conflict-free chromatic guarding of orthogonal polygons with holes* or CFSC-H for short. We first give an algorithm that we call CFSC-H-ThreeColors that uses three colors, thus proving an upper bound on the chromatic number for orthogonal polygons with holes.We also define a special class of polygons for which two colors are sufficient for CFSC-H, and give an algorithm to achieve such an assignment of colors.

4.1 Three colors are sufficient

Let P be an orthogonal polygon with holes (see Figure 7 for an example), and let $\mathbb{S}$ be the set of all maximal monotonous south-facing chains in P. Note that the chains in $\mathbb{S}$ belong to the outer boundary of P as well as the holes of P. By Lemma 1, a guard assigned to each of the south-facing edges, including the south-facing edges on the holes, covers the entire polygon P. Therefore, our goal is to assign a guard to each of the south-facing edges in a conflict-free manner.

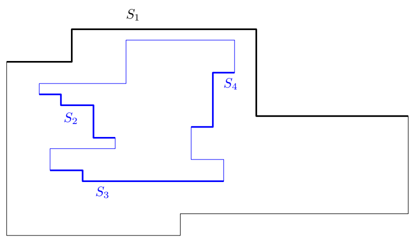


Figure 7: An orthogonal polygon with an orthogonal hole shown in blue; outer boundary has one and the hole has three maximal monotonous south-facing chains.

Since we are allowing one more color to be used than CFSC, a straightforward idea could be to follow a similar techniques to Algorithm CFSC-TwoColors from the previous section. However, that may not be possible. Take the polygon P in Figure 7 as an example.

Suppose that we first assign guards to S_3, which puts S_2 and S_4 in Q. After assigning guards to S_2, the queue Q contains S_4 and S_1. Now when we assign guards to S_4, we either have to put a duplicate entry for S_1 in the Q, or retrieve S_1 from Q to update the color constraints and then put it back again. In either case, searching the queue Q for an entry for a specific chain increases the time complexity of the algorithm. Therefore, we need some tool to obtain an efficient algorithm for CFSC-H. In this regard, we introduce the notion of *relationship graph*.

Definition 2 (Relationship graph) *The relationship graph $G_{\mathbb{S},P}$ of P is a directed graph whose nodes are the chains of $\mathbb{S}$ such that for each pair of chains $S, S' \in \mathbb{S}$, G has a directed edge from S to S' for each upward connection of S that is in the visibility region of S'. See Figure 8 for an example.*

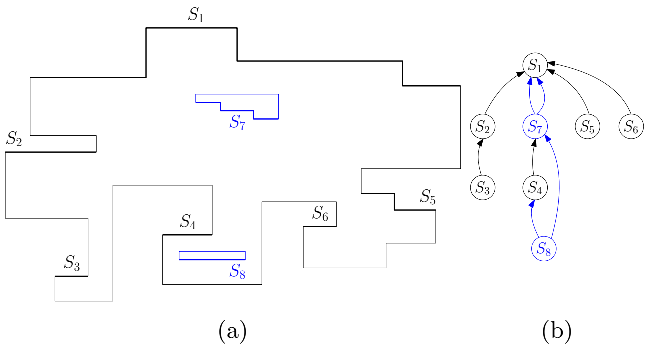


Figure 8: (a) Set $\mathbb{S}$ of maximal monotonous south-facing chains of orthogonal polygon P with holes, the holes are blue. (b) The relationship graph $G_{\mathbb{S},P}$.

We observe some properties of $G_{\mathbb{S},P}$.

Lemma 5 *$G_{\mathbb{S},P}$ is a directed acyclic graph (DAG) where each node of G has at most two outgoing edges.*

Proof. Since P is connected and simple, by Definition 2, there cannot be a directed cycle. Since a chain can have at most two upward connections, the number of outgoing edges of a node of G must be at most two. $\square$

We can build the relationship graph $G_{\mathbb{S},P}$ of P using the trapezoidal map $\mathcal{T}$ of the chains of $\mathbb{S}$ as in the previous section. For each *upward connection* e of each $S \in \mathbb{S}$, we traverse the search structure $\mathcal{D}$ associated with $\mathcal{T}$ to find the edge e' and the associated chain S' just above e. Let v be incident to edge e in S. We then add the edge (S, S') with the label (e, e') in $G_{\mathbb{S},P}$. Building $\mathcal{T}$ and $\mathcal{D}$ takes expected $O(n \log n)$ time using the incremental randomized algorithm in [7], and finding the edge above each endpoint takes $O(\log n)$ time. Therefore, the relationship graph G can be built from $\mathcal{D}$ in $O(n \log n)$ time.

We now describe Algorithm CFSC-H-ThreeColors. Since $G_{\mathbb{S},P}$ is a DAG by Lemma 5, we obtain a topological ordering of the nodes of the graph [6] in $O(n)$ time. We then assign guards to the chains of $\mathbb{S}$ according to topological ordering. For any chain $S \in \mathbb{S}$, one of these cases holds. **Case 1.** S **does not have any upward connections.** We place a blue camera at all edges e_i, where i is odd, and a red camera when i is even. **Case 2.** S **has only one upward connection.** Let the label of the upward connection edge be (e, e') where e belongs to S. Without loss of generality, assume that $c(e')$ is red. We then color e blue, and on either side of e, we assign red and blue cameras alternatingly to the rest of the edges of S. **Case 3.** S **has two upward connections.** Let the labels of the upward connections be (e_1, e'_1) and (e_q, e'_q) where e_1 and e_q are edges of S. Without loss of generality, assume that e'_1 is colored red. If e'_q is also red, we color $e_1, \ldots, e_q$ with colors blue and green alternatingly. If e'_q has a different color than e'_1, say blue, we assign the colors green and red to $e_1, \ldots, e_q$ starting with green. Then, we will not have conflict whether q is odd or even.

We now prove the correctness and time complexity of the algorithm. See the appendix for the complete proof.

Theorem 6 CFSC-H-ThreeColors *computes a conflict-free chromatic guarding of an orthogonal polygon P in $O(n \log n)$ time using three colors.*

Proof Sketch. The correctness follows from the fact that the topological ordering of the chains would ensure that any chain S with an upward connection to another chain S' would be colored after S. The calculation of running time is similar to Theorem 4.

4.2 Special case: monotonous rectangular holes

We describe a restricted class of orthogonal polygons, where the boundary of each hole is a rectangle and the order of the holes inside the polygon is monotone with respect to either x-axis or y-axis; see Figure 9(a). We call this restricted class of polygons *orthogonal polygons with monotonous rectangular holes.* Without loss of generality, we assume that the order of the holes is X-monotone. We give an algorithm that we call CFSC-H-TwoColors to obtain a CFSC-H using two colors.

Let $\mathbb{S}$ and $\mathbb{H}$ be the sets of maximal monotonous south-facing chains from the outer boundary of P and from the holes of P. As in Algorithm CFSC-TwoColors, we build the trapezoidal map $\mathcal{T}$ with $\mathbb{S}$ and $\mathbb{H}$, and from it we build the data structure $\mathcal{L}$. However, when checking the search structure of $\mathcal{T}$ for the edges above a chain $H \in \mathbb{H}$, we add all the chains that are above H.

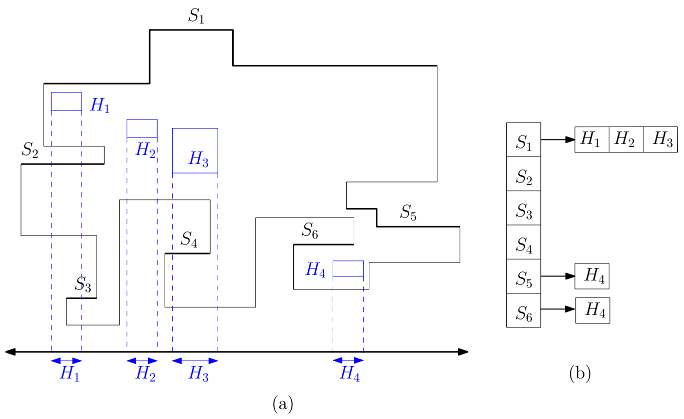


Figure 9: (a) An orthogonal polygon with rectangular X-monotone holes; the projection of the hole boundaries on the X-axis are nonoverlapping. (b) List of holes under each chain ordered from left to right according to the holes' projections on the X-axis.

We need another data structure to keep track of holes that are *directly under* a chain $S \in \mathbb{S}$. We consider the list $\mathcal{A}$ in Figure 9(b), where the holes directly under a chain S are listed from left to right. We can populate $\mathcal{A}$ at the same time as $\mathcal{L}$; and later sort each individual list of holes for each chain in $\mathbb{S}$ in ascending order of the x-coordinates of the holes.

Algorithm CFSC-H-TwoColors uses the same idea as Algorithm CFSC-TwoColors. For the first chain S chosen, we start from the leftmost edge e_1 of S. We assign blue to e_1, red to e_2, and continue this way until we reach a hole H_1 under S. Let $e', e'' \in S$, where $e' = e''$ may hold, be the two edges intersected by the two vertical edges of hole H when extended in the direction of positive y-axis. We assign the same color to all the edges of S from e' to e'', and assign the opposite color to e' to the top edge and the bottom edge of H_1. We then continue as before until we reach the next hole H_2 and follow the same procedure.

After coloring S and all the holes under it, we put the entries for all the uncolored chains in $\mathbb{S}$ that are influenced by the coloring of S in the queue Q. Since all the holes under S have already been colored, no holes will be added to Q at this step. Then for each hole H_i under S, we check if a chain S' influenced by it is already in Q. If S' is in Q, that means that it is also influenced by S. Then both S and S' must be above H_i and influenced by the guard of the north-facing edge e_h of H_i. Then we can assign the same color to all edges of S and S' that are in the visibility region of e_h. So we remove the entry for S' from Q and add a new entry that applies the opposite color of e_h to the visible edges of S'. Since the holes are X-monotone, no holes would be added to Q at this stage also.

We then remove the chain at the front of Q and apply the same procedure. We keep dequeuing chains from Q and assigning guards to them until Q is empty. Note that since the holes under a chain are colored at the time of coloring the chain, no hole would be added to Q at any point. Figure 10 describes some scenarios that may occur. Note that, in the case where e'' belong to a different chain $S_j \neq S_i$ (as in S_5 and S_6 for the hole H_4 in Figure 9), the guards on the top edge of H will see some edges of S_j. When coloring S_j, we propagate the opposite color of H on S_j from where the visibility of the top edge of H ends.

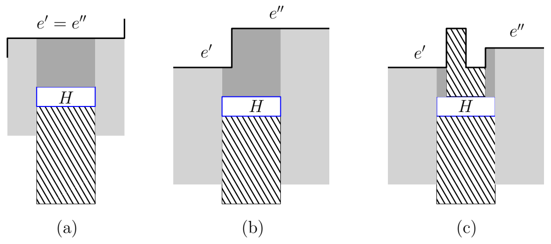


Figure 10: Different scenarios when assigning colors to the hole; the dark gray regions are covered by one blue (black patterned) and one red (solid light gray) guard: (a) $e' = e''$, (b) e', e'' are adjacent, and (c) there are edges between e' and e''.

The following theorem summarizes the time complexity and correctness of the above algorithm.

Theorem 7 *Algorithm* CFSC-H-TwoColors *computes a CFSC-H of an orthogonal polygon P with monotonous rectangular holes in $O(n^2)$ time using two colors.*

Proof. Building the data structure $\mathcal{L}$ and $\mathcal{A}$ takes $O(n \log n)$ time each by Theorem 4. Sorting all lists in $\mathcal{A}$ takes $O(n \log n)$ time in total. Since the holes are colored at the time of coloring the chains, the coloring is done in $O(n)$ time. However, when adding chains influenced by a hole, we have to look for them in Q. This may take $O(n)$ time. Therefore, the total running time of the algorithm is $O(n^2)$. □

5 Conclusion

We have given an $O(n \log n)$ time algorithm for CFSC with two colors for orthogonal polygons without holes and showed that the bound is tight. We have given an $O(n \log n)$ time algorithm for CFSC-H with three colors for polygons with holes, while a special case requires two colors. The question of whether two colors are sufficient for the general case of CFSC-H remains open. In our algorithms, the guards are placed only on horizontal edges. It would be interesting to investigate whether less guards are needed when placed on vertical and horizontal edges.

References

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

[2] P. Bhattacharya, S. K. Ghosh, and B. Roy. Vertex Guarding in Weak Visibility Polygons. In *CALDAM*, pages 45–57, 2015.

[3] P. Bhattacharya, S. K. Ghosh, and B. Roy. Approximability of guarding weak visibility polygons. *Discrete Applied Mathematics*, 228:109–129, 2017.

[4] T. Biedl, T. M. Chan, S. Lee, S. Mehrabi, F. Montecchiani, and H. Vosoughpour. On guarding orthogonal polygons with sliding cameras. In *WALCOM: Algorithms and Computation: 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29–31, 2017, Proceedings*, pages 54–65. Springer, 2017.

[5] O. Çağırıcı, S. K. Ghosh, P. Hliněný, and B. Roy. On conflict-free chromatic guarding of simple polygons. In *COCOA*, 2019.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001.

[7] M. de Berg, O. Cheong, M. Kreveld, and M. Overmars. *Computational Geometry, Algorithms and Applications.* Springer-Verlag, 3rd edition, 2008.

[8] M. de Berg, S. Durocher, and S. Mehrabi. Guarding monotone art galleries with sliding cameras in linear time. *Journal of Discrete Algorithms*, 44:39–47, 2017.

[9] S. Durocher, O. Filtser, R. Fraser, A. D. Mehrabi, and S. Mehrabi. Guarding orthogonal art galleries with sliding cameras. *Comput. Geom.*, 65:12–26, 2017.

[10] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100:238–245, 2006.

[11] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability Results for Guarding Polygons and Terrains. *Algorithmica*, 31:79–113, 2001.

[12] L. H. Erickson and S. M. LaValle. An art gallery approach to ensuring that landmarks are distinguishable. In *Robotics: science and systems*, volume 7, pages 81–88, 2012.

[13] S. Fisk. A short proof of Chvátal's watchman theorem. *J. Comb. Theory, Ser. B*, 24:374, 1978.

[14] M. Ghodsi, A. Maheshwari, M. N. Baygi, J.-R. Sack, and H. Zarrabi-Zadeh. α-visibility. *Comput. Geom. Theory Appl.*, pages 435–446, 2014.

[15] C. Iwamoto and T. Ibusuki. Computational Complexity of the Chromatic Art Gallery Problem for Orthogonal Polygons. In *WALCOM*, pages 146–157, 2020.

[16] M. J. Katz and G. Morgenstern. Guarding Orthogonal Art Galleries with Sliding Cameras. *Int. J. Comput. Geometry Appl.*, 21:241–250, 2011.

[17] J. King and D. G. Kirkpatrick. Improved Approximation for Guarding Simple Galleries from the Perimeter. *Discrete & Computational Geometry*, 46:252–269, 2011.

[18] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

[19] J. Urrutia. Chapter 22 - Art Gallery and Illumination Problems. In *Handbook of Computational Geometry*, pages 973–1027. Elsevier, 2000.

Appendix

Lemma 1. *A sliding camera on each south-facing edge of $\mathbb{S}$ collectively guards the entire polygon P.*

Proof. From any point $p \in P$ if we draw a vertical line upward, the first edge e that the line intersects must be south-facing. Then e must belong to one of the chains in $\mathbb{S}$. Therefore, each point of P is covered by an edge of $\mathbb{S}$, and thus putting a sliding camera on each of these edges will cover the whole polygon. $\square$

Theorem 3. *Algorithm* CFSC-MONOTONE *computes a CFSC of an X-monotone polygon P without holes in $O(n)$ time using only two colors, which is optimal.*

Proof. By Lemma 1, a sliding camera on each edge of S will guard the whole polygon. Since we are assigning alternating colored guards for adjacent edges of S, each point of P will have at least one unique colored guard. The guards can be assigned by a walk along the boundary of P, which takes $O(n)$ time. By Theorem 2, the lower bound for the number of colors needed for the guards is also two. Therefore, the number of colors used in this algorithm is optimal. $\square$

Theorem 4. *Algorithm* CFSC-TWOCOLORS *computes a CFSC of an orthogonal polygon P without holes in $O(n \log n)$ time using two colors.*

Proof. We first prove that the algorithm assigns guards to all the south-facing edges of P using only two colors. It is easy to see that the first chain S considered by the algorithm gets a conflict-free coloring with two colors. Now, if we remove the visibility region of S from P, we get disjoint sub-polygons $P_1, P_2, \ldots, P_q$ for some $q < n$. For each P_i, $1 \leq i \leq q$, only one endpoint of one south-facing edge has been assigned a color. Since the corresponding chain S' is placed in Q before any other chains in that sub-polygon, S" gets a conflict-free coloring with two colors. We then remove the visibility region of S' and prove the claim inductively for all the $S_i \in \mathbb{S}$.

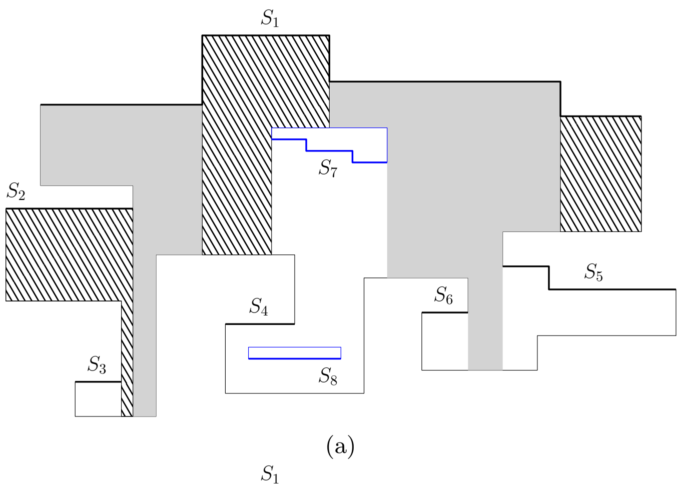
We now prove the running time of the algorithm. We assume that P is input as the sequence of vertices $v_1, \ldots, v_n$ in clockwise order. Then we can find the chains in $\mathbb{S}$ by walking around the polygon. We represent each chain $S \in \mathbb{S}$ by a tuple (v_i, v_j), where v_i is the first vertex and v_j is the last vertex of S on the walk. The south-facing edges in S can easily be calculated from the indices of the end vertices of S as $e_i, e_{i+2}, \ldots, v_{j-1}$. Therefore, assigning colors to all the south-facing edges takes $O(n)$ time. Building the trapezoidal map $\mathcal{T}$, searching the structure $\mathcal{D}$, and then populating the data structure $\mathcal{L}$ from $\mathcal{D}$ takes $O(n \log n)$ time. Therefore, the total time required is $O(n \log n)$. $\square$

Theorem 6. *Algorithm* CFSC-H-THREECOLORS *computes a conflict-free chromatic guarding of an orthogonal polygon P in $O(n \log n)$ time using three colors.*

Proof. The correctness follows from the fact that the topological ordering of the chains would ensure that any chain S with an upward connection to another chain S' would be colored after S.

We assume that the outer boundary of P is given as a clockwise sequence of vertices on it, and the hole boundaries are given as counterclockwise sequence of vertices on them. Building the trapezoidal map $\mathcal{T}$ takes expected $O(n \log n)$ time, and building the graph G from the search structure takes $O(n \log n)$ time. We obtain a topological ordering of the chains of $\mathbb{S}$ from the DAG G in $O(n)$ time. We then color the chains in $O(n)$ time in total. Therefore, the total time complexity of the algorithm is $O(n \log n)$. $\square$

Figure 11 shows how Algorithm CFSC-H-THREECOLORS works by showing the first two steps.



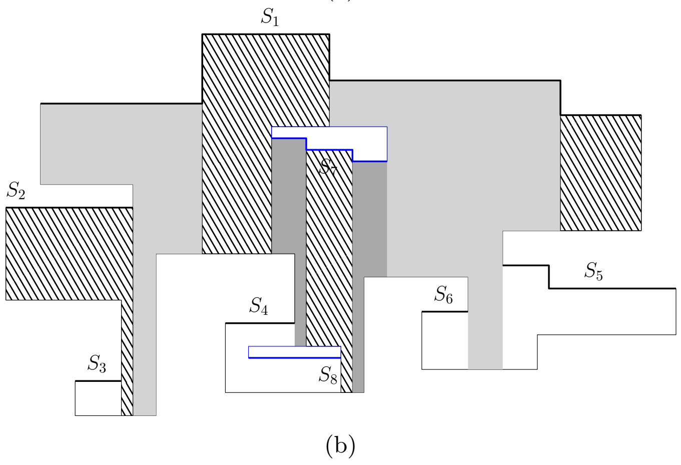


Figure 11: A conflict-free chromatic guarding of the polygon in Figure 8 with three colors. (a) After coloring S_1 and S_2. (b) After coloring S_7.

City Guarding with Cameras of Bounded Field of View

Ahmad Biniaz[*] Mohammad Hashemi[†]

Abstract

We study two problems related to the city guarding and the art gallery problems.

1. Given a city with k rectangular buildings, we prove that $3k+1$ cameras of $180°$ field of view are always sufficient to guard the free space (the ground, walls, roofs, and the sky). This answers a conjecture of Daescu and Malik (CCCG, 2020).

2. Given k orthogonally convex polygons of total m vertices in the plane, we prove that $\frac{m}{2}+k+1$ cameras of $180°$ field of view are always sufficient to guard the free space (avoiding all the polygons). This answers another conjecture of Daescu and Malik (Theoretical Computer Science, 2021).

Both upper bounds are tight in the sense that there are input instances that require these many cameras. Our proofs are constructive and suggest simple polynomial-time algorithms for placing these many cameras.

1 Introduction

Fixed cameras are common devices that are being used to monitor streets and buildings in cities. These cameras usually monitor the ground and walls. Due to an increasing use of drones and other flying objects, monitoring the entire space (including the ground, walls, roofs, and sky) is becoming crucial. The problem of monitoring the entire space with minimum number of cameras is usually referred to as the *city guarding* problem in computational geometry.

To the best of our knowledge the problems related to guarding cities were first introduced by Bao et. al [2]. They introduced three different versions of the problem where the goal is to guard (1) only the roofs of the buildings, (2) the walls of the buildings and the ground, and (3) the roofs, walls, and the ground. This latter version is called "city guarding".

In the city guarding problem we should take into account many factors such as the city's layout, buildings' orientation, and the cameras' field of view. These factors usually led to different variations of the city guarding problem.

In this paper we study a version of the city guarding problem that is introduced by Daescu and Malik [5]: *Given k pairwise disjoint rectangular-base buildings, find a minimum number of cameras that guard the city such that (i) each camera is a half-sphere with $180°$ field of view and infinite range, and (ii) each camera is placed at a corner on top of the roof of a building in a direction orthogonal to a wall.*

According to Bao et al. [2] the city guarding problem can be interpreted as a 2.5-dimensional version of the well-studied art gallery problem. In the standard art gallery problem we are given a simple polygon and the goal is to place the minimum number of guards/cameras to cover the entire polygon [8]. In other words, each point of the polygon is visible by some guard. A point p is said to be *visible* by a guard g if the line segment pg lies inside the polygon. The art gallery problem and its variations have been well-studied in recent years [12]. The variations usually enforce constraints on the shape of the polygon, the existence of holes, the shape of holes, the orientation of holes, locations of guards, guards' field and range of vision, to name a few. The city guarding problem has the same flavor as the art gallery problem with rectangular holes.

The city guarding also has the same flavor as a free-space illuminating problem, studied by Blanco et al. [3], in which the input consists of pairwise disjoint rectangles in the plane and the goal is to place minimum number of lights at the corners of the rectangles to light up the free space (the entire plane minus the the rectangles).

2 Related Works and Results

In this section we focus only on results that are directly related to the city guarding problem. There is a rich literature for the art gallery problem for which we refer the reader to [1, 3, 4, 7, 8, 9, 10, 11].

Bao et al. [2] studied the city guarding problem for k rectangular-base buildings that are orthogonal (to the xy-axis) and for cameras with $360°$ field of view. Recall that the cameras should be placed at top corners of buildings. They showed that $\lfloor\frac{2(k-1)}{3}\rfloor+1$ guards are always sufficient and sometimes necessary to guard the roofs. They also showed that $k+\lfloor\frac{k}{4}\rfloor+1$ guards are sufficient to guard walls and ground. For the city guarding (roofs, walls, the ground) they showed the sufficiency of

[*]School of Computer Science, University of Windsor, abiniaz@uwindsor.ca

[†]School of Computer Science, University of Windsor, hashem62@uwindsor.ca

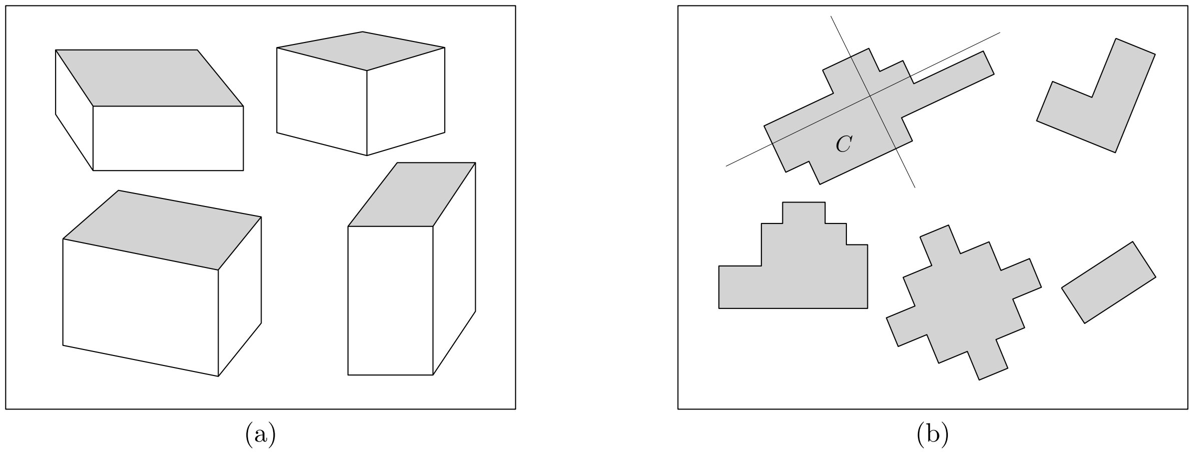


Figure 1: (a) A city with rectangular buildings. (b) Orthogonally convex polygons.

$k + \lfloor\frac{k}{2}\rfloor + 1$ guards.

Recently, Daescu and Malik [5] studied the city guarding problem for cameras with 180° field of view. They proved that $2k + \lfloor\frac{k}{4}\rfloor + 4$ cameras are sufficient to guard axis-aligned buildings. For arbitrary oriented buildings they gave an example that requires $3k + 1$ cameras for any $k \geq 1$. They conjectured that $3k + 1$ cameras are also sufficient. See Figure 1(a) for an example of arbitrary-oriented rectangular buildings.

In a companion paper, Daescu and Malik [6] studied another problem of the same flavor; guard free space formed by orthogonally convex polygons. Given k pairwise disjoint orthogonally convex polygons with total m vertices, the goal is to place cameras of 180° field of view to guard the free space and the boundaries of the polygons (cameras should be placed at corners of polygons and orthogonal to its sides). An *orthogonal polygon* is a polygon whose edges are orthogonal to each other (not necessarily orthogonal to the xy-axis). An orthogonal polygon is *orthogonally convex* if its intersection with any line orthogonal to its edges is either empty or a single line segment; see for example polygon C in Figure 1(b). Daescu and Malik show that for axis-aligned polygons $\frac{m}{2} + \lfloor\frac{k}{4}\rfloor + 4$ cameras are always sufficient and for arbitrary-oriented polygons $\frac{m}{2} + k + 1$ cameras are sometimes necessary for any $k \geq 1$ and any valid m. They conjectured that $\frac{m}{2} + k + 1$ cameras are also sufficient. See Figure 1(b) for an example of arbitrary-oriented orthogonally convex polygons.

2.1 Our Contributions

We prove both conjectures of Daescu and Malik [5, 6] that $3k + 1$ cameras are sufficient to guard arbitrary-oriented rectangular buildings, and $\frac{m}{2} + k + 1$ cameras are sufficient to guard arbitrary-oriented orthogonally convex polygons. Our proofs are constructive and suggest polynomial-time algorithms for finding these many guards. The two proofs share some similarities in the sense that both partition the free space into convex regions and then provide an upper bound for the number of these regions. We explain our proof for rectangular buildings first as it is easier to explain. Then we give a short description of how to generalize it for monotone orthogonal polygons.

3 City Guarding

In this section we present our algorithm for the city guarding problem. The following lemma, borrowed from [5], implies that to guard the entire space it suffices to guard roofs, walls, and the ground. Therefore in the algorithm we focus on guarding roofs, walls, and the ground.

Lemma 1 (Daescu and Malik [5]) *If in a city the roofs, walls, and the ground are guarded by a set of cameras, then every point in the aerial space of the city is visible by a camera.*

Recall that the city consists of k arbitrary-oriented buildings with rectangular basis, and that the cameras have 180° field of view and should be placed at corners on top of the roofs orthogonal to a wall. (We clarify that a camera could be placed in such a way that it sees the roof of the building, as in Figure 3.)

Daescu and Malik [5] gave an example which requires $3k + 1$ cameras. This example is given in Figure 2. Each building B_{i+1} is higher than the building B_i. They conjectured that the bound $3k + 1$ is tight.

We show how to guard the city with at most $3k + 1$ cameras, and thus proving the conjecture of [5]. We project the buildings onto the plane to obtain rectangles (in dimension 2). Then we guard the the rectangles (representing roofs), their sides (representing walls), and the space between them (representing the ground).

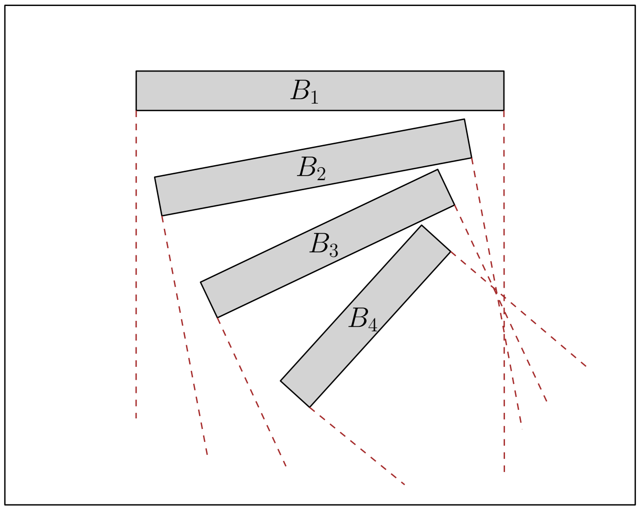


Figure 2: A city with $k = 4$ buildings that needs $3k + 1$ guards; borrowed from [5].

By Lemma 1, this would give a guarding of the city in dimension 3.

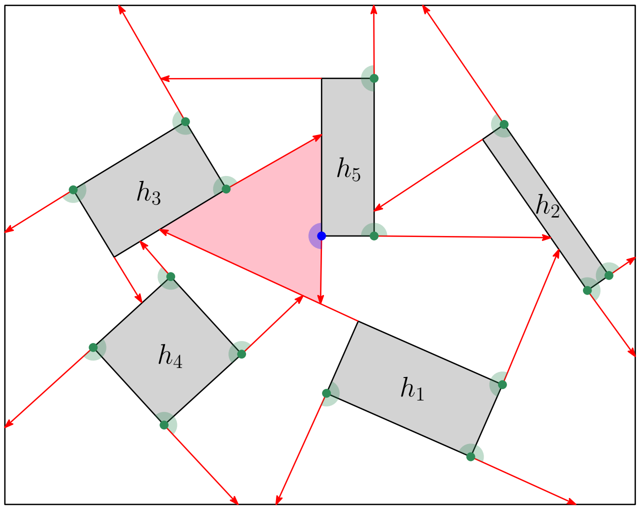


Figure 3: A city with $k = 5$ buildings. The sides are extended in order h_1, h_2, h_3, h_4, h_5. The pink area is a bad region. The green marks are corner guards and the blue mark is an edge guard.

We start by projecting the buildings vertically into the plane; this is a typical first step for problems of this type, see e.g. [2, 5]. Thus we obtain k pairwise disjoint rectangles in the plane. We may assume without loss of generality that the k rectangles lie in a bigger rectangle called P. One can think of P as a polygon and of rectangles as holes. Thus after this projection, each building becomes a hole in P and each wall becomes a side of some hole. One can think of this as an instance of the art gallery problem consisting of a polygon with rectangular holes.

Our next step is to guard P by cameras with 180° field of view. This would give (after lifting the rectangles back to their original height) a guarding of walls and the ground. As we will see later, our placement of cameras would guard the roofs as well.

Let $h_1, h_2, \ldots, h_k$ denote the rectangular holes ordered arbitrarily. For each h_i in this order, we extend the sides of h_i in counterclockwise direction and stop as soon as reaching another hole, an extension of a previous side, or the boundary of P; see Figure 3. Each extension is essentially a *directed line segment* whose initial point is a hole corner. These extensions partition P into some regions that we denoted $R_1, R_2, \ldots$; notice that we exclude the holes.

Lemma 2 *Each region R_i is convex.*

Proof. The region R_i is an intersection of a set of quadrants (which are convex). Each quadrant is defined by extensions of two adjacent sides of the same hole. Since the intersection of any set of convex objects is known to be convex, the region R_i is convex. □

Lemma 3 *The number of regions $R_1, R_2, \ldots$ is $3k+1$.*

Proof. We define a plane graph $G = (V, E)$ as follows. The vertex set V consists of the corners of the holes and the intersection points of the extended sides. We refer to them by *corner* and *intersection* vertices, respectively. The edges in E are formed by the sides of the holes, the extensions of sides, and the boundary of P.

We claim that each vertex of G has degree 3, and thus G is 3-regular. Each corner vertex is incident to two sides of a hole and an extension, thus has degree 3. Each intersection vertex is incident to an extension and two segments obtained from the intersected segment, and thus has degree 3. Degenerate cases are rather easy to handle, for example if two extensions hit a segment at the same point p, then we treat p as two vertices of degree 3 instead of one vertex of degree 4.

The number of corner vertices is $4k$. Each extension (of a side of a hole) defines an intersection vertex. Thus the number of intersection vertices is the same as the total number of sides of holes, which is $4k$. Therefore $|V| = 8k$. Since the sum of the vertex degrees in any graph is twice the number of edges and G is 3-regular, we have the following equality,

$$2|E| = 3|V|.$$

Therefore,

$$|E| = \frac{3|V|}{2} = \frac{3 \cdot 8k}{2} = 12k.$$

Let F be the set of faces of G, which includes the holes, the outerface (exterior of P), and the regions

$R_1, R_2, \ldots$. Using Euler's formula for connected planar graphs, we have

$$|F| = |E| - |V| + 2 = 12k - 8k + 2 = 4k + 2.$$

Excluding the outerface and the k holes, the number of regions $R_1, R_2, \ldots$ is $3k + 1$. □

Lemma 4 *Each region R_i contains a corner of a hole on its boundary.*

Proof. Recall the extensions of $h_1, \ldots, h_k$ in this order. Observe that the boundary of R_i contains (parts of) some extensions. Consider the last extension that was added to the boundary of R_i, or say, closes the region R_i. The entire directed line segment that defines this extension is part of the boundary of R_i. The initial point of this directed line segment is a corner of a hole. □

By Lemma 4 each region R_i has a hole corner on its boundary. If the boundary of R_i has a 90° angle at some corner, then we call it a *good region*, and otherwise a *bad region*; see Figure 3.

Camera Placement: Take any region R_i. If R_i is a bad region then let c be an arbitrary corner on the boundary of R_i. We place a camera at c facing towards the interior of R_i and perpendicular to the boundary segment of R_i containing c. We call this camera an *edge guard*—it lies on an edge of R_i. If R_i is a good region then let c be the lowest (i.e. with the smallest y-coordinate) corner at which the boundary of R_i has angle 90°. We place a camera at c facing towards the interior of R_i and perpendicular to the clockwise boundary segment at c (which is the extension at c). We call this camera a *corner guard*—it lies on a corner of R_i.

Since R_i is convex (by Lemma 2) the camera that is placed on the boundary of R_i covers the entire interior of R_i. Since we place exactly one camera for each region R_i, (i) all regions $R_1, R_2, \ldots$ are guarded, and (ii) the number of cameras is equal to the number of regions R_i which is $3k+1$ by Lemma 3. Therefore we have guarded the polygon P by $3k + 1$ guards. As discussed earlier, this gives a guarding of walls and the ground in the city.

We claim that our camera placement, also guards the roofs. Observe that for each hole h it holds that one of its corners is the lowest corner of angle 90° on the boundary of some good region R_i. Notice that such a lowest corner of R_i is uniquely defined by h. The camera that is placed at that corner (perpendicular to the extended side), guards the roof of h. The following theorem summarizes our result of this section.

Theorem 5 *Given k arbitrary-oriented rectangular-base buildings, we can guard the entire space (the ground, walls, roofs, and the sky) with at most $3k + 1$ cameras of 180° field of view that are placed at top corners of buildings orthogonal to a wall. The bound $3k+1$ is the best achievable.*

4 Guarding Orthogonally Convex Polygons

In this section we present our algorithm for guarding the free space formed by orthogonally convex polygons. Recall that the scene consists of k arbitrary-oriented orthogonally convex polygons, and that the cameras have 180° field of view and should be placed on corners of polygons orthogonal to a side. We may assume without loss of generality that the k polygons lie in a rectangular polygon called P. The free space, that we need to guard, is the interior of P minus the k given polygons.

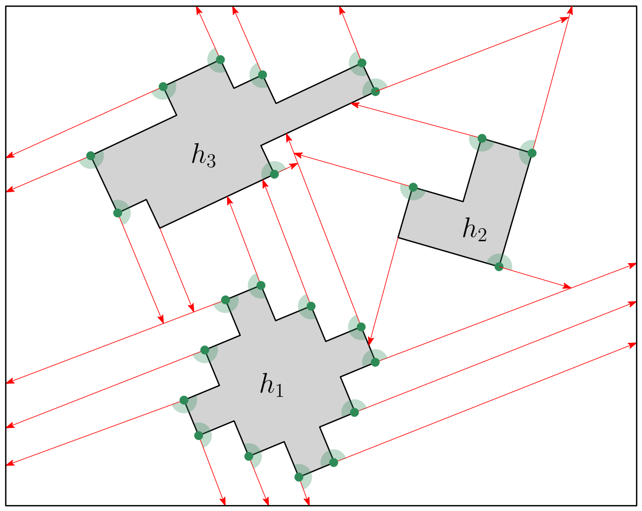


Figure 4: Three orthogonally convex polygons in the plane. The green marks are corner guards.

Similar to our algorithm for the city guarding in previous section we extend the sides of the polygons to partition the free space into convex region and then use one camera for each region. Let $h_1, h_2, \ldots, h_k$ denote the polygons in an arbitrary order. For each h_i in this order, we extend the sides of h_i in counterclockwise direction and stop as soon as reaching another polygon, an extension of a previous side, or the boundary of P. We only extend the sides whose extensions do not intersect the interior of h_i; see Figure 4. Thus we extend one side for every convex corner of a polygon. These extensions partition the free space into some regions that we denoted $R_1, R_2, \ldots$.

By an argument similar to that of Lemma 2 we can show that each R_i is convex.

By an argument similar to that of Lemma 3 we can show that the number of regions R_i is $\frac{m}{2} + k + 1$. We define a 3-regular plane graph $G = (V, E)$ as before. Among all corners, we only introduce vertices for convex ones. By a simple counting argument one can show that the total number of convex corners is $c = \frac{m}{2} + 2k$; see also [6]. Thus the number of vertices of G is $2c$, one vertex for each convex corner and one vertex for its extension. Thus $|V| = 2c = m + 4k$. Since the graph is 3-regular, the total degree is $3|V| = 3m + 12k$, which is equal to $2|E|$. Hence $|E| = \frac{3m}{2} + 6k$. Thus, for the number of faces we get

$$|F| = \left(\frac{3m}{2} + 6k\right) - (m + 4k) + 2 = \frac{m}{2} + 2k + 2.$$

Excluding the outerface and the k holes, the number of regions R_i is $\frac{m}{2}+k+1$. Similar to Lemma 4 we can show that each R_i has a corner on its boundary. We classify the regions by *good* and *bad* and then place cameras on the corners (one camera for each R_i) similar to our placement in the previous section. This would guard the free space with $\frac{m}{2} + k + 1$ cameras. The following theorem summarizes our result in this section.

Theorem 6 *Given k pairwise disjoint arbitrary-oriented orthogonally convex polygons of total m vertices in the plane, we can guard the entire free space with at most $\frac{m}{2} + k + 1$ cameras of $180°$ field of view that are placed at the corners of the polygons orthogonal to a side. The bound $\frac{m}{2} + k + 1$ is the best achievable.*

Remark. It is easily seen that the algorithm of this section can be generalized to guard cities with buildings that have orthogonally convex bases. In fact, the city guarding in the previous section is a special case of this problem where $m = 4k$.

References

[1] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. The art gallery problem is $\exists\mathbb{R}$-complete. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 65–73, 2018.

[2] Lichen Bao, Sergey Bereg, Ovidiu Daescu, Simeon Ntafos, and Junqiang Zhou. On some city guarding problems. In *International Computing and Combinatorics Conference*, pages 600–610. Springer, 2008.

[3] Gregoria Blanco, Hazel Everett, Jesus Garcia Lopez, and Godfried Toussaint. Illuminating the free space between quadrilaterals with point light sources. In *Proceedings of Computer Graphics International, World Scientific,* 1994.

[4] Vasek Chvátal. A combinatorial theorem in plane geometry. In *Journal of Combinatorial Theory*, *Series B* 18, pages 39–41, 1975

[5] Ovidiu Daescu and Hemant Malik. City guarding with limited field of view. In *Proceedings of 32nd Canadian Conference on Computational Geometry (CCCG)*, pages 300-311, 2020.

[6] Ovidiu Daescu and Hemant Malik. New bounds on guarding problems for orthogonal polygons in the plane using vertex guards with halfplane vision. In *Theoretical Computer Science*, 882, Pages 63-76, 2021.

[7] Frank Hoffmann. On the rectilinear art gallery problem. In *International Colloquium on Automata, Languages, and Programming*, pages 717–728. Springer, 1990.

[8] Joseph O'Rourke. *Art gallery theorems and algorithms.* Oxford University Press Oxford, 1987

[9] Joseph O'Rourke. Galleries need fewer mobile guards: A variation on Chvátal's theorem. In *Geometriae Dedicata*, 14, pages 273–283, 1983.

[10] Csaba D. Tóth. Art galleries with guards of uniform range of vision. In: *Computational Geometry*, Volume 21, pages 185–192, 2002.

[11] Csaba D. Tóth. Art gallery problem with guards whose range of vision is 180. In *Computational Geometry: Theory and Applications*, 17, pages 121–134, 2000.

[12] Jorge Urrutia. Art gallery and illumination problems. In *Handbook of computational geometry*, pages 973–1027. Elsevier, 2000.

On the complexity of embedding in graph products*

Therese Biedl† David Eppstein‡ Torsten Ueckerdt§

Abstract

Graph embedding, especially as a subgraph of a grid, is an old topic in VLSI design and graph drawing. In this paper, we investigate related questions concerning the complexity of embedding a graph G in a host graph that is the strong product of a path P with a graph H that satisfies some properties, such as having small treewidth, pathwidth or treedepth. We show that this is NP-hard, even under numerous restrictions on both G and H. In particular, computing the row pathwidth and the row treedepth is NP-hard even for a tree of small pathwidth, while computing the row treewidth is NP-hard even for series-parallel graphs.

1 Introduction

Layered treewidth, layered pathwidth, and row treewidth are structural parameters of graphs that have played a central role in recent developments in graph product structure theory. (The original graph product structure theorem was proved by Dujmovíc et al. [7]; see also [6, 9, 19] for improvements and related results.) Testing whether a graph has layered pathwidth ≤ 1 is NP-complete [2]. In this work we ask analogous questions about the computational complexity of the *row treewidth* of a graph G, the minimum possible treewidth of a graph H such that G is a subgraph of the strong product $H \boxtimes P_\infty$ where P_∞ is a 1-way infinite path:

- Is it NP-hard to compute the row treewidth?
- Is it NP-hard to test whether a planar graph has row treewidth 1, its smallest nontrivial value?
- How complicated must G be for these problems to be hard? Are they easier for planar graphs?

Row treewidth can be naturally generalized to other product forms for H. For example, the *row pathwidth* of a graph G is the smallest possible pathwidth of a graph H such that G is a subgraph of $H \boxtimes P_\infty$, and similarly one can define the *row treedepth* or *row simple treewidth* or *row simple pathwidth*. The above questions could be asked for any of these parameters.

These questions have a geometric flavor coming from the grid-like graph products they concern. They are special cases of subgraph isomorphism, which is hard even under strong restrictions on both G and the host graph [17]. Our answers are that these problems are indeed hard, even for very simple graphs. It is NP-hard to test whether a tree of bounded pathwidth has row pathwidth one, and the same holds for row simple pathwidth and row treedepth. Row treewidth is trivial for trees, but it is NP-hard to test whether series-parallel graphs of bounded degree and bounded pathwidth have row treewidth one. Under the small set expansion conjecture (a strengthening of the unique games conjecture from computational complexity theory), row treewidth, row pathwidth, layered treewidth, and layered pathwidth are hard to approximate with constant approximation ratio. We provide a few positive results as well: Testing embeddability in $P \boxtimes P$ (a grid with diagonals) is polynomial for caterpillars, and testing embeddability in $P \square P$ (a grid) is polynomial for planar graphs of bounded treewidth and bounded face size.

1.1 Definitions

A *tree decomposition* of a graph G is a tree T whose vertices (‘*bags*’) are labeled with subsets of vertices of G. Each vertex must belong to bags forming a connected subtree of T, and each edge of G must have both endpoints included together in at least one bag. If T is a path, it forms a *path decomposition*. The *width* of the decomposition is the size of the largest bag, minus one. The *treewidth* of G is the smallest width of a tree decomposition of G, and the *pathwidth* is the smallest width of a path decomposition. A tree decomposition is w-*simple* if each set of w vertices belongs to at most two bags. The *simple pathwidth [simple treewidth]* of G is the smallest w such that G has a w-simple path [tree] decomposition of width $\leq w$. The *treedepth* of a graph G is the smallest height of a rooted tree T on the vertices of G such that every edge of G connects an ancestor-descendant pair in T.

For connected graphs with at least one edge, these width parameters have minimum value one. The graphs with treewidth one are trees. The graphs with treewidth two are the *series-parallel graphs* and their subgraphs. The graphs with pathwidth one are not just paths, but

*Work initiated at the Workshop on Graph Product Structure Theory (BIRS21w5235) at the Banff International Research Station, November 21-26, 2021.

†David R. Cheriton School of Computer Science, University of Waterloo. Supported by NSERC.

‡Department of Computer Science, University of California, Irvine. Research supported in part by NSF grant CCF-2212129.

§Institute of Theoretical Informatics, Karlsruhe Institute of Technology

caterpillars: trees whose non-leaf vertices form a path called the *spine*. (The leaves of a caterpillar are called *legs*.) The graphs with simple pathwidth one are paths. The graphs with treedepth one are *stars*, graphs $K_{1,\ell}$ for integer ℓ. To avoid having to specify a specific number of vertices, it is convenient to let $P_\infty = \langle p_0, p_1, \dots \rangle$ denote a *ray*, a one-way infinite path, to let C_∞ denote the caterpillar with infinite-length spine and infinitely many legs at each spine-vertex, and to let S_∞ be a star with infinitely many degree-1 vertices.

The *strong product* of two graphs $G \boxtimes H$ has a vertex (u_i, v_j) for each pair of a vertex u_i in G and a vertex v_j in H, and an edge connecting two pairs (u_i, v_j) and $(u_{i'}, v_{j'})$ when u_i and $u_{i'}$ are either adjacent in G or identical, and v_i and $v_{i'}$ are either adjacent in H or identical. For instance, the strong product of two paths is a *king's graph*, the graph of moves of a chess king on a chessboard whose rows and columns are indexed by the vertices of the paths (see also Fig. 1).

A *layering* of a graph G is a partition of the vertices into sets $L_0, L_1, \dots$ such that for any edge the endpoints are in the same or in consecutive sets. It can be understood as a representation of a graph G as a subgraph of $P \boxtimes K$ for a path P and a complete graph K; the layers of the layering are the subsets of vertices in this product coming from the same vertex of the path. Layered tree decompositions and path decompositions of a graph consist of a tree or path decomposition of the graph, together with a layering. Their width is the size of the largest intersection of a bag with a layer, minus one. The layered treewidth [8, 18] or layered pathwidth [2] of G is the minimum width of such a decomposition. Instead, the *row treewidth* or *row pathwidth* of G is the minimum treewidth or pathwidth of a graph H for which G is a subgraph of $P \boxtimes H$ for some path P. Intuitively, row treewidth and row pathwidth restrict the notion of layered treewidth and layered pathwidth by requiring each layer to have the same decomposition. These concepts are not equivalent: the layered treewidth of any graph G is at most its row treewidth plus one, but there exist graphs with layered treewidth one and arbitrarily large row treewidth. A similar separation occurs also between layered pathwidth and row pathwidth [5]. We can similarly define layered simple treewidth/simple pathwidth/treedepth and row simple treewidth/simple pathwidth/treedepth; to our knowledge these parameters have not been studied previously.

We show that the following problems are NP-hard:

- RowSimplePathwidth: Given a graph G and an integer k, does G have row simple pathwidth at most k? We will show that this is NP-hard even for $k = 1$, where it becomes the question whether G is a subgraph of $P_\infty \boxtimes P_\infty$, i.e., the king's graph, which is why we also call this problem KingGraphEmbedding. See Section 2 and Appendix A.
- RowPathwidth: Given a graph G and an integer k, does G have row pathwidth at most k? We will show that this is NP-hard even for $k = 1$, where it becomes the question whether G is a subgraph of $C_\infty \boxtimes P_\infty$. See Section 3.
- RowTreewidth: Given a graph G and an integer k, does G have row treewidth at most k? We will show that this is NP-hard even for $k = 1$, where it becomes the question whether G is a subgraph of $T \boxtimes P_\infty$ for some tree T. See Section 4 and Appendix B.
- RowTreedepth: Given a graph G and an integer k, does G have row treedepth at most k? We will show that this is NP-hard even for $k = 1$, where it becomes the question whether G is a subgraph of $S_\infty \boxtimes P_\infty$. See Appendix C.

It is helpful to introduce some notation for the strong product $H \boxtimes P_\infty$. Recall that P_∞ is a ray $\langle p_0, p_1, \dots \rangle$. For any vertex $v \in H$, the *P-extension* is the set of vertices $\langle (v, p_0), (v, p_1), \dots \rangle$. For any vertex $(v, p_i) \in H \times P_\infty$, the *H-projection* is the vertex v and the *P-projection* is the vertex p_i. These concepts naturally extend to edges and paths. Inspired by the case $H = P_\infty$ (where $H \boxtimes P_\infty$ is the king's graph) we define the following *edge-orientations*: An edge vw of $H \boxtimes P_\infty$ is *horizontal* if v, w have the same H-projection, *vertical* if they have the same P-projection, and *diagonal* otherwise. Since we expand along a path P_∞, every vertex of $H \boxtimes P_\infty$ has only two incident horizontal edges.

We will occasionally also study the *Cartesian product* $H \square P_\infty$ of two graphs, which is the same as the strong product except diagonals are omitted.

2 Grid embeddings

In this section we study KingGraphEmbedding. This problem is closely related to GridEmbedding, the question whether a given graph G is a subgraph of the rectangular grid $P_\infty \square P_\infty$. GridEmbedding is old and well-studied since at least the 1980s due to its connections to VLSI design. Bhatt and Cosmadakis showed in 1987 [3] that GridEmbedding is NP-hard even for trees of pathwidth 3 (the pathwidth was not studied explicitly by the authors, but can be verified from the construction). Gregori [13] expands their proof to binary trees. Both proofs use a technique later called the "logic engine" by Eades and Whitesides [10]. Recently, Gupta et al. [15] strengthened the result to trees of pathwidth 2.

Theorem 1 (Gupta et al. [15]). GridEmbedding *is NP-hard even for a tree of pathwidth 2.*

The reductions in [3, 15] can be modified to work for KingGraphEmbedding. Even easier is to use the following general-purpose transformation.

Define T_1 and T_2 to be the trees shown in Fig. 2, formed by subdividing one edge of $K_{1,1}$ and $K_{1,4}$ respec-

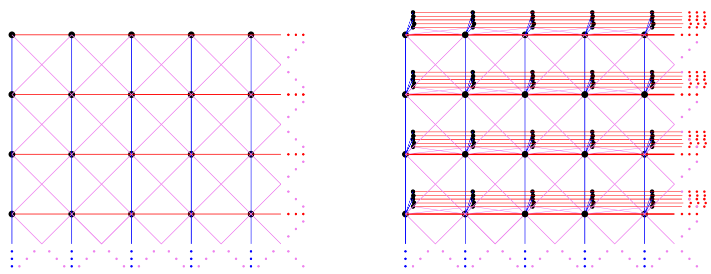
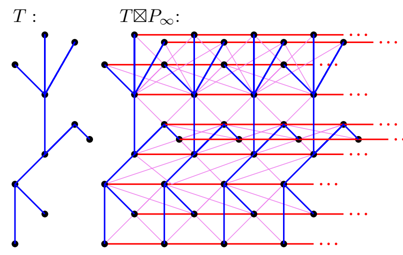



Figure 1: The graphs $P_\infty \boxtimes P_\infty$, $C_\infty \boxtimes P_\infty$, and $T \boxtimes P_\infty$ for a tree T.

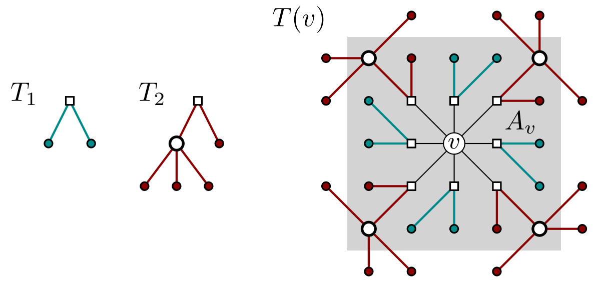


Figure 2: The trees T_1, T_2, and $T(v)$ in Observation 2.

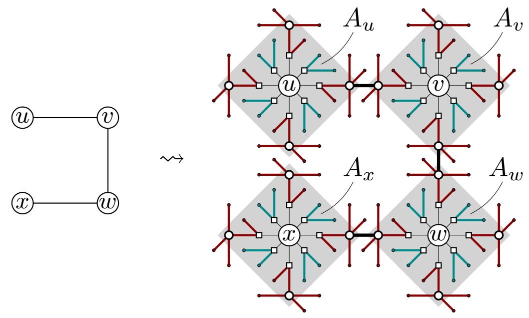


Figure 3: If $G \subset P_\infty \Box P_\infty$, then $G' \subset P_\infty \boxtimes P_\infty$. (We show a 45° rotation of $P_\infty \boxtimes P_\infty$.)

tively. For a given vertex v, define $T(v)$ toto be a tree rooted at v with eight children: four copies each of T_1 and T_2, connected to v at their degree-2 vertices. The following is not hard to verify (see Appendix A):

Observation 2. *Let G be a simple graph. Form G' by replacing each vertex v in G by a new tree $T(v)$, and connecting a degree-4 vertex in $T(u)$ with a degree-4 vertex in $T(v)$ for each edge uv in G. Then $G \subset P_\infty \Box P_\infty$ if and only if $G' \subset P_\infty \boxtimes P_\infty$.*

The transformation clearly maintains a tree. Replacing each vertex v by a tree $T(v)$ of radius 3 increases the pathwidth by at most 3, so applying the transformation to the tree of Gupta et al. [15] gives the following.

Corollary 3. KingGraphEmbedding *is NP-hard, even for a tree of pathwidth at most* 5.

In fact, one can easily adapt the reduction of Gupta et al. [15] to show NP-hardness of KingGraphEmbedding even for a tree of pathwidth 2; see Fig. 5 in Appendix A for an illustration.

2.1 Caterpillars

On the other hand, for pathwidth 1 (i.e., caterpillars), we can solve KingGraphEmbedding in linear time.

Theorem 4. *For any caterpillar G the following are equivalent.*

(1) $G \subset P_\infty \boxtimes P_\infty$

(2) G can be embedded in $P_\infty \boxtimes P_\infty$ such that all spine edges are diagonal

(3) For every subpath Q of the spine of G we have $\sum_{v \in V(Q)} \deg(v) \leq 6|V(Q)| + 2$.

Proof. (1)$\Longrightarrow$ (3): Assume that G is a caterpillar that is a subgraph of $H = P_\infty \boxtimes P_\infty$. Let Q be any fixed subpath of the spine of G. Clearly, for any vertex $v \in P_\infty \boxtimes P_\infty$ the neighbourhood $N_H(v)$ of v in graph H satisfies $|N_H(v)| \leq 8$ and for any edge $uv \in P_\infty \boxtimes P_\infty$ we have $|N_H(u) \cap N_H(v)| \geq 2$. As caterpillar G contains no triangles, for any two adjacent vertices x, y in G we have $N_G(x) \cap N_G(y) = \emptyset$. Hence

$$\sum_{v \in V(Q)} \deg(v) \leq 8|V(Q)| - 2|E(Q)| = 6|V(Q)| + 2.$$

(3) $\Longrightarrow$ (2): Assume that $G = (V, E)$ is a caterpillar, say with spine $\langle v_1, \ldots, v_k \rangle$. The vertices of $P_\infty \boxtimes P_\infty$ naturally corresponds $\mathbb{N} \times \mathbb{N}$, where (p_i, p'_j) is mapped to (i, j). We embed the spine of G along the main diagonal, i.e., place v_i at (i, i) for $i = 1, \ldots, k$. Then, we proceed along the spine from v_1 to v_k, always placing the next leg at v_i at the positions (x, y) adjacent to (i, i) with $x + y$ as small as possible. Let us say that v_i is *free* if two leaves at v_i are embedded successfully at $(i-1, i)$ and $(i, i-1)$, respectively. In particular, the first vertex with degree at least 4 is always free. (We assume there exists such a vertex, otherwise G clearly can be embedded.)

Assume that this embedding procedure fails to find a suitable position for a leaf at v_j for some $j \in [k]$. Let $i \leq j$ be the largest index such that v_i is free, and $Q = \langle v_i, \ldots, v_j \rangle$ be the subpath of the spine of G from v_i to v_j.

Observe that $\deg_G(v_i), \deg_G(v_j) \geq 4$ and further that for $A = \{(i,i),\ldots,(j,j)\}$, there is a vertex in $V(Q) \cup N_G(Q)$ on each of the $5(j-i)+9$ points in $A \cup N(A)$ in $P_\infty \boxtimes P_\infty$. With $|V(Q) \cup N_G(Q)| = \sum_{v \in V(Q)} \deg(v) - (j-i) + 1$, it follows that

$$\begin{aligned} & |V(Q) \cup N_G(Q)| > |A \cup N(A)| \\ \Leftrightarrow & \sum_{v \in V(Q)} \deg(v) - (j-i) + 1 > 5(j-i) + 9 \\ \Leftrightarrow & \sum_{v \in V(Q)} \deg(v) > 6(j-i) + 8 = 6(j-i+1) + 2 \\ & \qquad = 6|V(Q)| + 2, \end{aligned}$$

which implies that G does not satisfy (3).

(2) $\implies$ (1): This is immediate. ☐

Corollary 5. KingGraphEmbedding *can be solved in linear time for n-vertex caterpillars.*

Proof. Let G be a caterpillar with spine $\langle v_1, \ldots, v_k \rangle$, $k \leq n$. Using (3) in Theorem 4, G admits *no* embedding into $P_\infty \boxtimes P_\infty$ if and only if the sequence $(\deg(v_i)-6)_{i \in [k]}$ has a contiguous subsequence whose sum is at least 3. Finding such a subsequence is the MaximumSubarray problem and can be solved in time $O(k)$ [14]. ☐

3 Row pathwidth

Now we consider the row pathwidth, and show that testing whether the row pathwidth is 1 is NP-hard. This is the same as asking whether a given graph G is a subgraph of $C_\infty \boxtimes P_\infty$. We also consider the related problem of embedding in $C_\infty \square P_\infty$. Both problems are easily shown NP-hard using another observation concerning how graph transformations affect embeddability.

Observation 6. *Let G be a simple graph, and for $k \in \{4,6\}$ let G'_k be the result of adding (at any original vertex v of G) $\max\{0, k - \deg(v)\}$ leaves that are adjacent to v. Then*

- *$G \subset P_\infty \square P_\infty$ if and only if $G'_4 \subset C_\infty \square P_\infty$.*
- *$G \subset P_\infty \boxtimes P_\infty$ if and only if $G'_6 \subset C_\infty \boxtimes P_\infty$.*

Proof. The forward direction is obvious: If G is such a subgraph, then take the embedding of G in the grid, and use the P-extensions of k legs at each spine-vertex of C_∞ to place the added leaves at each vertex v.

For the other direction, observe that all vertices on P-extensions of legs of C_∞ have degree at most 3 in $C_\infty \square P_\infty$, and degree at most 5 in $C_\infty \boxtimes P_\infty$. We constructed G'_k (for $k \in \{4,6\}$) such that the vertices of G have degree k, so they must be placed on the P-extension of a spine-vertex. If we set π to be the spine of C_∞, therefore G is embedded in $\pi \square P_\infty$ (respectively $\pi \boxtimes P_\infty$) as desired. ☐

Theorem 7. *It is NP-hard to test whether a tree is a subgraph of $C_\infty \boxtimes P_\infty$. It is also NP-hard to test whether a tree is a subgraph of $C_\infty \square P_\infty$. Both results hold even for trees with constant maximum degree and pathwidth 3.*

Proof. By the discussion after Corollary 3, testing whether $G \subset P_\infty \boxtimes P_\infty$ is NP-hard, even for a tree with pathwidth 2. Convert G into G' using Observation 6 with $k = 6$. This preserves a tree, increases the pathwidth by at most 1, and the maximum degree is 8. Also $G \subset P_\infty \boxtimes P_\infty$ if and only if $G' \subset C_\infty \boxtimes P_\infty$, which proves the first claim. The second claim is similar, using Theorem 1 and Observation 6 with $k = 4$. ☐

Corollary 8. RowPathwidth *is NP-hard, even for trees of bounded degree and pathwidth, and even if we only want to know whether the row pathwidth is 1.*

4 Row treewidth

We now sketch why computing row treewidth NP-hard. (The full proof is in Appendix B.)

Theorem 9. *It is NP-hard to test whether a graph G is a subgraph of $T \boxtimes P_\infty$ for some tree T, even for a series-parallel graph G.*

Our reduction from NAE-3SAT uses the *logic engine* of Eades and Whitesides [10], a general approach to convert an instance I of NAE-3SAT into a geometric shape that can be realized in the plane if and only if I has a solution. We only explain this for our particular problem. We may assume that I has a clause $x_n \vee \overline{x_n}$, because we can add this without affecting existence of a solution. We first construct a graph G_0 and designate some edges as horizontal/vertical. (Fig. 6 in the appendix shows G_0, while Fig. 4 shows the graph derived from it.) Start with the *frame* (orange) which consists of three paths connecting two vertices t, b; the *middle path* has $H := m + 2n + 1$ vertical edges, while the two *outer paths* have $2n$ horizontal, H vertical, and then another $2n$ horizontal edges each. Next add the *armature of x_i* (light/dark cyan) for each variable x_i, which consists of two paths that attach at the vertices of the middle path at distance i from t and b. The paths are assigned to literals x_i and $\overline{x_i}$ and consist of $2n+1-2i$ horizontal edges at both ends with $H - 2i$ vertical edges inbetween. The middle m rows of our drawing are called the *clause-rows* and assigned to one clause each. Finally we attach *flags* (green). Namely, at the vertex where the armature of literal ℓ_i intersects the row of c_j, we attach a leaf (via a horizontal edge) if and only if ℓ_i does *not* occur in c_j. This finishes the construction of G_0.

Next we add more vertices and edges that force edge-orientations to be what we specified for G_0. First, "triple the width": insert a new column before and after every column that we had in our drawing of G_0, subdivide

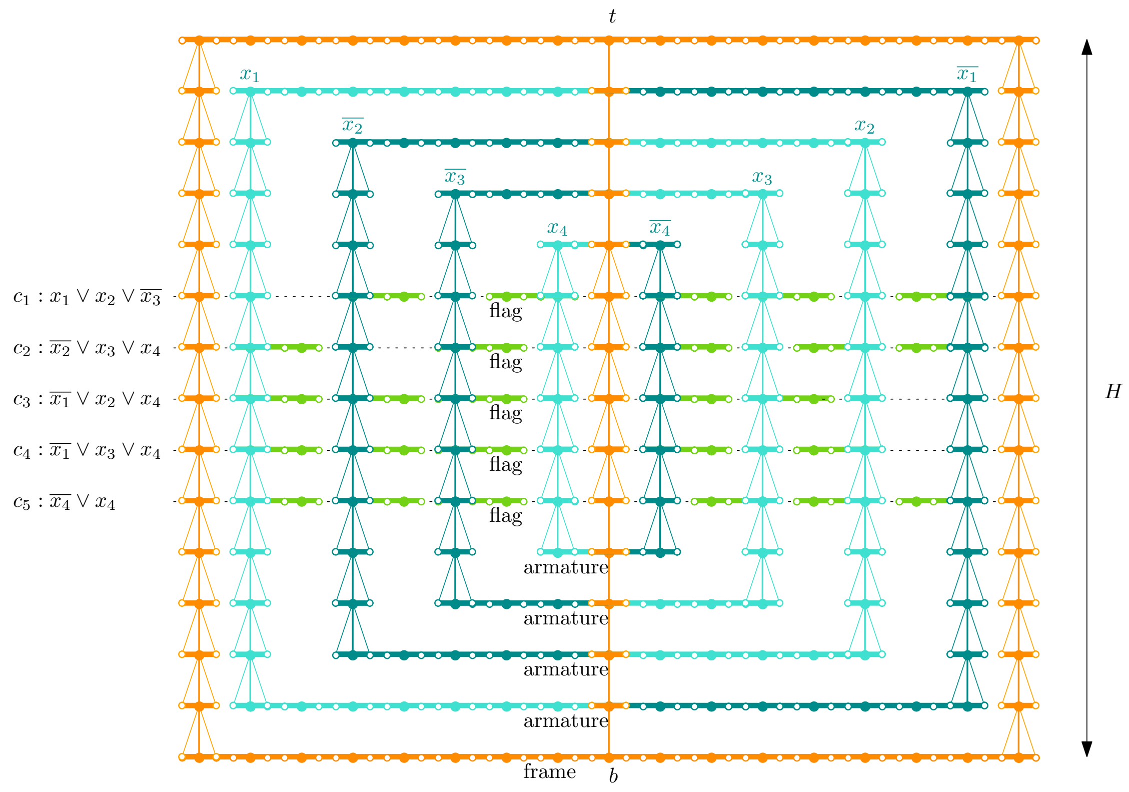

t
x_1
$\overline{x_1}$
$\overline{x_2}$
x_2
$\overline{x_3}$
x_3
x_4
$\overline{x_4}$
$c_1 : x_1 \vee x_2 \vee \overline{x_3}$
$c_2 : \overline{x_2} \vee x_3 \vee x_4$
$c_3 : \overline{x_1} \vee x_2 \vee x_4$
$c_4 : \overline{x_1} \vee x_3 \vee x_4$
$c_5 : \overline{x_4} \vee x_4$
flag
flag
flag
flag
flag
armature
armature
armature
armature
frame
b
H

Figure 4: The reduction for row-treewidth. Bold edges indicate an attached $K_{2,5}$. Vertices of G_0 are solid.

each horizontal edge of G, and for every vertex v with k incident horizontal edges, add $2-k$ new leaves connected via horizontal edges. (New vertices are hollow in Fig. 4.) Next *add an arrow-head* at some vertical edges vw. Assuming v is below w, this means adding the edges (v_ℓ, w) and (v_r, w), where v_ℓ, v_r are the two neighbours of v adjacent to it via horizontal edges. We add arrow-heads at all vertical edges except those of the middle path where an armature or outer path attaches at the lower endpoint. Call the result G'. Finally we turn G' into G by *adding a* $K_{2,5}$ at every horizontal edge uv, i.e., adding five new vertices that are adjacent to both u and v. (To avoid clutter we do not show $K_{2,5}$ in Fig. 4, but indicate it with a bold edge.) Call the resulting graph G, and verify that it is indeed a series-parallel graph.

One can argue (see the appendix) that if G is embedded in $T \boxtimes P_\infty$ for some tree T, then all edges with attached $K_{2,5}$ must be horizontal. This in turn forces that G' is actually embedded within $P_\infty \boxtimes P_\infty$ (this is the hardest part). The arrow-heads force the edges at which they are attached to be vertical, and with a counting-argument therefore the embdding of G' implies an embedding of G_0 in $P_\infty \square P_\infty$ where the designated orientations are respected. This is (with the standard logic engine argument) easily seen to be equivalent to the NAE-3SAT instance having a solution.

The graph in our construction has maximum degree 16 and pathwidth $O(1)$, so computing the row treewidth remains NP-hard even under these restrictions.

Corollary 10. RowTreewidth *is NP-hard, even for series-parallel graphs of bounded degree and pathwidth, even if we only want to know whether the row treewidth is 1.*

A similar construction shows that testing whether $G \subset T \square P_\infty$ for some tree T is also NP-hard. Namely, use the same construction (G_0 to G' to G), except omit the diagonal edges and replace '$K_{2,5}$' by 'three paths of length 2'. This forces all horizontal edges to have the desired orientation in any embedding of G in $T \square P_\infty$. Argue as above that then G lies within $\pi \square P_\infty$ for a path π. Therefore any vertical edge uv must have this orientation, because both u, v have two incident horizontal edges. So this gives an embedding of G_0 in the grid that respects the given orientation, hence a solution to NAE-3SAT.

5 Inapproximability

The best approximation ratio known for a polynomial-time approximation algorithm for the treewidth is $O(\sqrt{\log w})$, where w is the treewidth [11]. No polynomial-time constant-factor approximation algorithms are known, and they are unlikely to exist, since such algorithms would violate a standard assumption in computational complexity theory, the small set expansion conjecture [20]. As we now show, the same hardness results apply to the approximation of row treewidth and row pathwidth:

Theorem 11. *If there exists an approximation algorithm for row treewidth, row pathwidth, layered treewidth, or layered pathwidth with approximation ratio ρ, then there exists an approximation algorithm for treewidth or pathwidth (respectively) with approximation ratio at most 3ρ. As a consequence, the small set expansion conjecture implies that ρ cannot be $O(1)$.*

Proof. Let G be a graph for which we wish to approximate the treewidth or pathwidth, let w be its treewidth or pathwidth, and form graph G^+ with treewidth or pathwidth $w + 1$ by adding a universal vertex to G. The universal vertex forces every layering of G^+ to use at most three layers. G^+ has a trivial layering with one layer and row treewidth or row pathwidth $w + 1$. Any other layering has row treewidth, row pathwidth, layered treewidth, or layered pathwidth at least $(w + 1)/3$, because it gives a tree decomposition for G^+ with bags that are the unions of bags in three layers. Therefore, any approximation for the row treewidth, row pathwidth, layered treewidth, or layered pathwidth of G^+ gives an approximation for the treewidth or pathwidth of G^+, and therefore of G, with approximation ratio increased by at most a factor of three. □

Note that the constructed graph G^+ is not necessarily planar. In fact, for planar graphs there are $O(1)$-approximation algorithms for the treewidth [16].

6 Outlook

In this paper, we proved that computing graph parameters such as the row pathwidth and row treewidth are NP-hard to compute, even under strong restrictions on the input graph. In fact, most of these restrictions rule out hopes for fixed-parameter tractability (or at least the possibility of finding polynomial-time algorithms in special situations). We do state here a few possibilities of situations where finding an embedding may be polynomial, but this mostly remains for future work:

- Given a graph with bounded radius, is it possible to solve RowTreewidth or RowPathwidth in polynomial time? In all our hardness constructions, the graph had radius $\Theta(n)$. Bounded radius forces any layering to use a bounded number of rows, so if the row treewidth or row pathwidth is also bounded, then the treewidth or pathwidth of the original graph must also be bounded, but it is not obvious how to take advantage of this in an algorithm.
  GridEmbedding *is* polynomial for graphs of bounded radius, because a graph can be embedded in a grid only if it has bounded maximum degree, and together with bounded radius this would imply bounded size, hence a constant-time algorithm.
- For the results for GridEmbedding ([3] and graph G_0 in the proof of Theorem 9) we very much needed the ability to change the embedding of the graph, so that we could flip armatures and flags. What is the status if the embedding is fixed? In particular, is testing whether a tree can be embedded in a grid NP-hard if the embedding of the tree is fixed, possibly similar to the results in [1]?

One could also ask for results for planar graphs with a fixed embedding where faces are small (e.g. triangulations). In all our constructions, some faces have degree $\Theta(n)$. Can we solve any of the problems (but especially KingGraphEmbedding) for triangulated planar graphs? This remains open, but we can make some progress if additionally also the treewidth is small.

Theorem 12. *Let G be a planar graph with treewidth t and a planar drawing Γ where all faces have degree at most Δ. Then we can test whether G can be embedded in the grid (in a way that respects embedding Γ) in time $O^*(n^{3(t+1)\Delta})$, i.e., in polynomial time if $t \cdot \Delta \in O(1)$.*

Proof. In 2013, the first author and Vatshelle [4] studied the PointSetEmbedding problem, where we are given a set of points S and a planar graph G, and we ask whether G has a planar straight-line drawing where all vertices are placed at points of S. They showed that if G has treewidth at most t and face-degree at most Δ, then PointSetEmbedding can be solved in $O^*(|S|^{1.5(t+1)\Delta})$ time. Their approach is to use a so-called carving decomposition of the dual graph, which results in a hierarchical decomposition of G into ever smaller subgraphs H (ending at one face) for which the *boundary* (the vertices of H that may have neighbours outside H) has small size. The main idea to solve PointSetEmbedding is then to do dynamic programming in this carving decomposition, and the parameter for the dynamic program is all possible embeddings of the boundary of H in the given point set S.

To adapt this algorithm to our situation, we need two changes. First, we fix the point set S to be the points of an $n \times n$-grid. (Clearly no bigger grid can be required.) In particular, we have $|S| = n^2$. Second, when considering possible embeddings of the boundary of H, we *only* consider such embeddings where this boundary is drawn along edges of the grid with diagonals. With this restriction, the same dynamic program will test whether a grid embedding exists in the desired time. □

Sadly this approach does not seem to generalize to our other embedding problems.

References

[1] Hugo A. Akitaya, Maarten Löffler, and Irene Parada. How to fit a tree in a box. *Graphs and Combinatorics*, 38(5):155, 2022. doi:10.1007/s00373-022-02558-z.

[2] Michael J. Bannister, William E. Devanny, Vida Dujmović, David Eppstein, and David R. Wood. Track layouts, layered path decompositions, and leveled planarity. *Algorithmica*, 81(4):1561–1583, 2019. doi:10.1007/s00453-018-0487-5.

[3] Sandeep Bhatt and Stavros Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Information Processing Letters*, 25(4):263–267, 1987. doi:10.1016/0020-0190(87)90173-6.

[4] Therese Biedl and Martin Vatshelle. The point-set embeddability problem for plane graphs. *Int. J. Comput. Geom. Appl.*, 23(4-5):357–396, 2013. doi:10.1142/S0218195913600091.

[5] Prosenjit Bose, Vida Dujmović, Mehrnoosh Javarsineh, Pat Morin, and David R. Wood. Separating layered treewidth and row treewidth. *Discrete Mathematics & Theoretical Computer Science*, 24(1):P18:1–P18:10, 2022. doi:10.46298/dmtcs.7458.

[6] Prosenjit Bose, Pat Morin, and Saeed Odak. An optimal algorithm for product structure in planar graphs. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022*, volume 227 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SWAT.2022.19.

[7] Vida Dujmovic, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020. doi:10.1145/3385731.

[8] Vida Dujmović, Pat Morin, and David R. Wood. Layered separators in minor-closed graph classes with applications. *J. Combinatorial Theory, Ser. B*, 127:111–147, 2017. doi:10.1016/j.jctb.2017.05.006.

[9] Zdeněk Dvořák, Tony Huynh, Gwenael Joret, Chun-Hung Liu, and David R. Wood. Notes on graph product structure theory. In Jan de Gier, Cheryl E. Praeger, and Terence Tao, editors, *2019-20 MATRIX Annals*, pages 513–533. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-62497-2_32.

[10] Peter Eades and Sue Whitesides. The logic engine and the realization problem for nearest neighbor graphs. *Theoretical Computer Science*, 169(1):23–37, 1996. doi:10.1016/S0304-3975(97)84223-5.

[11] Uriel Feige, Mohammadtaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[13] Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31(4):167–173, 1989. doi:10.1016/0020-0190(89)90118-X.

[14] David Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2(3):207–214, 1982. doi:10.1016/0167-6423(83)90015-1.

[15] Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi. Grid recognition: Classical and parameterized computational perspectives. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021*, volume 212 of *LIPIcs*, pages 37:1–37:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.37.

[16] Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. *Theor. Comput. Sci.*, 645:60–90, 2016. doi:10.1016/j.tcs.2016.06.040.

[17] Jirí Matousek and Robin Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discret. Math.*, 108(1-3):343–364, 1992. doi:10.1016/0012-365X(92)90687-B.

[18] Farhad Shahrokhi. New representation results for planar graphs. In *Proc. 29th European Workshop on Computational Geometry (EuroCG '13)*, pages 177–180, 2013. arXiv:1502.06175.

[19] Torsten Ueckerdt, David R. Wood, and Wendy Yi. An improved planar graph product structure theorem. *Electron. J. Comb.*, 29(2), 2022. doi:10.37236/10614.

[20] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. *J. Artificial Intelligence Research*, 49:569–600, 2014. doi:10.1613/jair.4030.

A Missing details from Section 2

We first give a proof of Observation 2: Any graph G can be modified into a graph G' such that G has an embedding in $P_\infty \square P_\infty$ if and only if G' has an embedding in $P_\infty \boxtimes P_\infty$.

Proof. The forward direction is obvious: If $G \subset P_\infty \square P_\infty$, then take the embedding, rotate it by 45° and stretch it such that neighboring grid vertices are $5\sqrt{2}$ units apart. Place this in $P_\infty \boxtimes P_\infty$ and verify that each $T(v)$ can be placed, and for each edge of G the two respective degree-4 can be connected as in Fig. 3.

For the other direction, assume G' has an embedding in $P_\infty \boxtimes P_\infty$. Observe that for any vertex v in G, the set $S_v = \{w \in V(G') \colon \operatorname{dist}(v,w) \leq 2\}$ has size $|S_v| = 1+8+16 = 25$, and occupies a 5×5 square area A_v in $P_\infty \boxtimes P_\infty$. For any edge $e = uv$ in G, the corresponding 5-path u-s_1-s_2-s_3-s_4-v must be embedded along five diagonals of $P_\infty \boxtimes P_\infty$ with the same slope. This holds as s_2 has four neighbors outside S_u (s_3 and three vertices of $T(u) \setminus S_u$) and thus must be on a corner of A_u; and symmetrically s_3 lies on a corner of A_v. Finally (s_2, s_3) must be diagonal (and have the same slope), otherwise there would not be six vertices of $P_\infty \boxtimes P_\infty$ that are outside $S_u \cup S_v$ but adjacent to s_2 or s_3. □

Next we sketch (in Fig. 5) how to take the specific tree from the NP-hardness construction from [15], and directly construct a tree T' of pathwidth 2 that has an embedding in $P_\infty \boxtimes P_\infty$ if and only if T has an embedding in $P_\infty \square P_\infty$. Thus KingGraphEmbedding is NP-hard even for trees of pathwidth 2.

B Row treewidth

We prove here Theorem 9: It is NP-hard to test whether a graph G is a subgraph of $T \boxtimes P_\infty$ for some tree T, even for a series-parallel graph G. We already sketched the construction in Section 4; we repeat the full construction here for ease of reading.

The reduction is from NAE-3SAT and uses the *logic engine* by Eades and Whitesides [10], which maps instances of NAE-3SAT to geometric gadgets called *frames*, *armatures* and *flags*, in such a way that the entire construct has an embedding in the plane if and only if the NAE-3SAT instance has a solution. The frame here has only one embedding (up to symmetry), while the armatures have some flexibility (and express whether a variable is true or false), and the flags are used to enforce that every clause has at least one true and at least one false variable. In our case we ask for an embedding in a graph, and therefore the gadgets are small subgraphs.

We first show NP-hardness of a closely related problem. Assume that with a graph G, we are also given labels 'hor' and 'ver' on some of its edges. We say that an embedding of G in $T \boxtimes P_\infty$ is *orientation-constrained* if the edges marked 'hor/ver' are horizontal and vertical, respectively. (Recall that horizontal/vertical means that the two endpoints of the edge have the same T-projection/P-projection.)

Claim 13. *Consider the following problem: 'Given a graph G with labels hor/ver on some edges, does it have an orientation-constrained embedding in $T \boxtimes P_\infty$ for some tree T?' This is NP-hard, even for a series-parallel bipartite graph G.*

Proof. Let I be an instance of NAE-3SAT with n variables and m clauses. We construct G and at the same time discuss possible orientation-constrained embeddings of G in the grid, see also Fig. 6. (Since we restrict *all* edges to be horizontal or vertical, it does not matter whether the grid includes the diagonals or not.) Start with the *frame* (orange in the figure) which consists of three paths connecting two vertices t, b; the *middle path* has $H := m + 2n + 1$ vertical edges, while the two *outer paths* have $2n$ horizontal, H vertical, and then another $2n$ horizontal edges each. An orientation-constrained embedding of the frame in the grid is unique up to symmetry. The middle m rows of this embedding are called the *clause-rows* and marked with one clause each.

Next we add the *armature of x_i* (light/dark cyan) for each variable x_i. This consists of two paths that attach at the vertices of the middle path at distance i from t and b. Each path consist of $2n + 1 - 2i$ horizontal edges at both ends with $H - 2i$ vertical edges inbetween. The paths are assigned to literals x_i and $\overline{x_i}$. An orientation-constrained embedding of frames and armatures in the grid is unique up to symmetry and up to horizontally flipping each armature; in particular the row of each vertex is unchanged over all such embeddings.

Finally we attach *flags* (green) at the intersections of armatures and clause-rows. Namely, at the vertex where the armature of literal ℓ_i intersects the row of c_j, we attach a leaf (via a horizontal edge) if and only if ℓ_i does *not* occur in c_j. For each flag we have the choice of whether to place it to the right or to the left of its attachment vertex, as long as this spot has not been used by a different flag already. Graph G is clearly series-parallel, because we can reduce it to an edge by deleting leaves and multiple edges and contracting degree-2 vertices. (We remind the reader of the following equivalent definitions of series-parallel graphs: (a) Connected graphs without a K_4-minor, (b) connected graphs of treewidth 2, (c) graphs obtained from an edge by attaching leaves and duplicating or subdividing edges, (d) connected graphs for which all 3-connected components contain at most three vertices.)

If I has a solution, then flip the armatures such that the left paths correspond to the literals of the solution. For each clause c_j there exists at least one true literal, hence there are at most $n - 1$ flags in the row of c_j

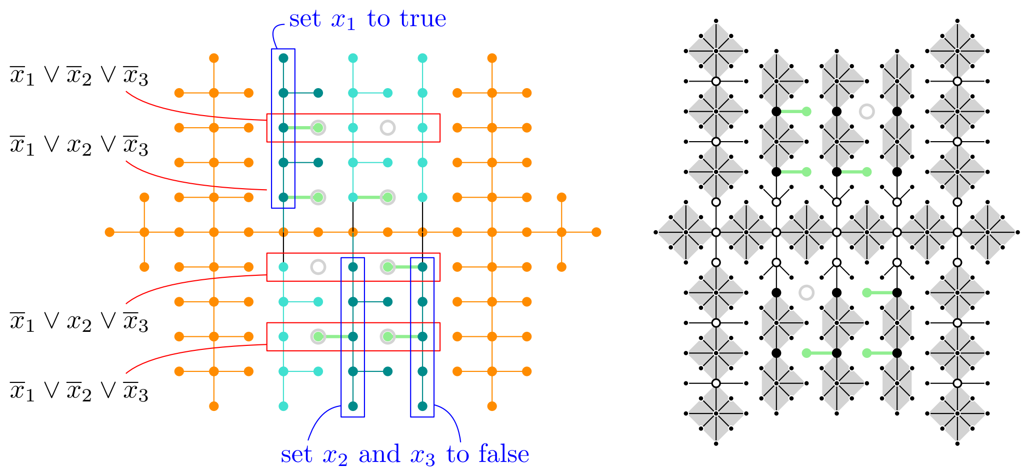

set x_1 to true
$\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$
$\overline{x}_1 \vee x_2 \vee \overline{x}_3$
$\overline{x}_1 \vee x_2 \vee \overline{x}_3$
$\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$
set x_2 and x_3 to false

Figure 5: Left: The tree of pathwidth 2 of Gupta et al. [15] for the NAE-SAT instance $\varphi = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$ in its GridEmbedding for solution $\{x_1 = \text{true}, x_2 = \text{false}, x_3 = \text{false}\}$. Right: A corresponding tree of pathwidth 2 in its corresponding king's graph embedding. ($P_\infty \boxtimes P_\infty$ is rotated 45°.)

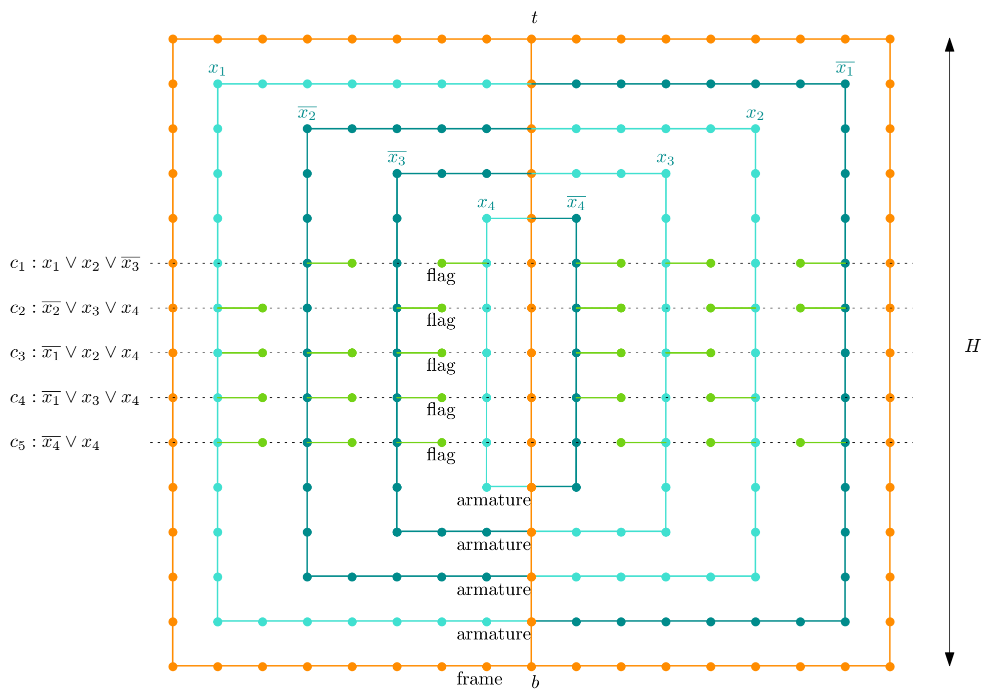

t
x_1
$\overline{x_1}$
$\overline{x_2}$
x_2
$\overline{x_3}$
x_3
x_4
$\overline{x_4}$
$c_1 : x_1 \vee x_2 \vee \overline{x_3}$
$c_2 : \overline{x_2} \vee x_3 \vee x_4$
$c_3 : \overline{x_1} \vee x_2 \vee x_4$
$c_4 : \overline{x_1} \vee x_3 \vee x_4$
$c_5 : \overline{x_4} \vee x_4$
flag
flag
flag
flag
flag
armature
armature
armature
armature
frame
b
H

Figure 6: The reduction for row-treewidth if we can fix the orientation of edges.

and left of the middle path; we can arrange them as to fit within the gaps. There also exists at least one false literal, hence at most $n-1$ flags in the row of c_j to the right of the middle path. So we can find an orientation-constrained embedding of G in the grid. Vice versa, if we have such an embedding, then taking the literals that are left of the middle path gives a solution to I because for each clause c_j there must be at most $n-1$ flags on each side of the middle path, so at least one literal is true and at least one literal is false.

So I has a solution if and only if G has an orientation-constrained embedding in the grid. To finish the NP-hardness, we must argue that any orientation-constrained embedding of G in $T \boxtimes P_\infty$ for some tree T actually must reside within a grid. To see this, let π be the path in T that corresponds to the T-projection of one outer path of the frame. Since the edge-orientations on the outer path are fixed, π has length H and connects the T-projections t', b' of t and b, so t', b' have distance H in T. We claim that the embedding of G actually resides within $\pi \boxtimes P_\infty$, i.e., for any vertex v of G the T-projection v' of v is on π. To show this, observe that we can find a path from t to v by walking through the frame, then (perhaps) an armature and then (perhaps) along a flag, and always only go downward. Similarly find a path from v to b that only goes downward. The combined walk σ_v from t to b via v uses exactly H non-horizontal edges. The T-projection σ'_v of σ_v connects t' to b' and has length H, which by uniqueness of paths in trees implies that $\sigma'_v = \pi$ contains v'. $\square$

To prove Theorem 9, we take the construction of Claim 13, but add more vertices and edges to obtain a graph G for which edge-orientations are forced in any embedding of G in $T \boxtimes P_\infty$.

So assume that we are given an instance I of NAE-3SAT. We may assume that one clause of I is $x_n \vee \overline{x_n}$, for if there is no such clause, then we can add it without affecting the solvability of I. Now let G_0 be the graph constructed for instance I as in the proof of Claim 13. As before, G_0 has a unique orientation-constrained embedded in the grid up to horizontal flipping of armatures and flags, so the y-coordinates of vertices are fixed. We call the vertices and edges of G_0 *original*.

As our next step, we "triple the width". Roughly speaking, we insert a new column before and after every column that we had in the drawing of G_0. Formally (and explained on the graph, rather than the drawing), subdivide every horizontal edge twice, and at any vertex v incident to k horizontal edges, attach $2-k$ leaves. All new edges are again required to be horizontal. See Fig. 4, ignoring bold lines and diagonal edges for now. The resulting graph G_1 likewise has an orientation-constrained embedding in the grid if and only if the NAE-3SAT instance has a solution. It also clearly is series-parallel since it is obtained from G_0 by subdividing edges and attaching leaves.

Next we obtain G_2 by *adding an arrow-head* at some vertical edges vw. Assuming v is below w, this means adding the edges (v_ℓ, w) and (v_r, w), where v_ℓ, v_r are the two neighbours of v adjacent to it via horizontal edges. We add arrow-heads at all vertical edges of G_1except those of the middle path where an armature or outer path attaches at the lower endpoint. The graph stays series-parallel since each arrow-head $\{v_\ell, v, v_r, w\}$ contains a cutting pair that separates it from the rest of the graph, so adding the edges of the arrow-heads does not affect whether there are non-trivial 3-connected components.

For the final modification we need a simple but crucial observation, which one proves by inspecting the neighbourhood of two adjacent vertices in $T \boxtimes P_\infty$ for all possible orientations of the edge between them.

Observation 14. *Let G be a graph embedded in $T \boxtimes P_\infty$ for some tree T. If uv is an edge of G for which u, v have at least five common neighbours, then uv must be horizontal.*

Thus, we turn G_2 into G by *adding a $K_{2,5}$* at every 'hor' edge uv of G_2, i.e., adding five new vertices that are adjacent to both u and v. This keeps the graph series-parallel and force uv to be horizontal in any embedding of G in $T \boxtimes P_\infty$. To avoid clutter we do not show $K_{2,5}$ in Fig. 4, but indicate it with a bold edge.

This ends the description of our construction. It should be straightforward to see that a solution to the NAE-3SAT instance I implies that G can be embedded in $C_\infty \boxtimes P_\infty$. Namely, we can embed G_0 in $\pi \boxtimes P_\infty$ where π is the spine of C_∞, subdivide each edge of P_∞ twice to embed G_1, realize the arrow-heads along diagonals, and finally use 5 legs at each vertex of π to embed the attached $K_{2,5}$'s on their P-extensions. Vice versa, assume that G is embeded in $T \boxtimes P_\infty$ for some tree T. We know that all bold edges must be horizontal. We also claim that if vw was a 'ver' edge of G_1 that received an arrow-head, then the orientation of vw in the embedding is vertical. To see this, assume that the arrow-head was $\{v_\ell, v, v_r, w\}$, with v_ℓ, v_r connected to v via horizontal edges. Then vw belongs to two triangles $\{v_\ell, v, w\}$ and $\{v_r, v, w\}$, and the two horizontal edges (v_ℓ, v) and (v_r, v) of these triangles share endpoint v. No two such triangles exist at a diagonal edge, and vw cannot be horizontal since the two horizontal edges at v are vv_ℓ and vv_r. So vw is vertical.

We claim that the embedding of G_2 in $T \boxtimes P_\infty$ actually resides within $\pi \boxtimes P_\infty$ for some path π, i.e., in a grid. This is argued almost exactly as in Claim 13. Let π be the T-projection of one outer path of G_0; since the orientations of the edges on the outer path is fixed π has length H. For any vertex v of G_2, we can find a walk σ_v from t to b via v that uses exactly H non-horizontal edges (they may now be diagonal). As before this implies that the T-projection of v is also in π, so the embedding of G_2 is within $\pi \boxtimes P_\infty$, i.e., the king's graph. As before, we can hence associate vertices of G_2 with points in $\mathbb{N} \times \mathbb{N}$, and speak of rows and columns of this embedding.

Since the orientations of edges on the outer paths are fixed, the drawing of the outer paths is fixed up to symmetry and spans $12n+3$ columns (including the space for the arrowheads). The rest of G_2 must lie inside the outer-paths, so in particular the row of clause $x_n \vee \overline{x_n}$ (which we call the *spacer-row*) has $12n+3$ points that could host vertices. But there are three paths of the frame, $2n$ armature-paths and $2(n-1)$ flags in this row, meaning $4n+1$ original vertices use the spacer-row. Since we tripled the width, all $12n+3$ possible points in the spacer-row are used in the embedding of G_2. Furthermore, the vertices in the spacer-row come as triplets connected by horizontal edges, with the middle vertex the original vertex. Up to a translation therefore all original vertices in the space-row have x-coordinate divisible by 3. This forces any original vertex w to have x-coordinate divisible by 3 as well, because we can get from w to an original vertex v in the spacer-row using only edges that must be vertical (due to an arrow-head) or horizontal (due to a $K_{2,5}$), and the horizontal parts have length divisible by 3. In consequence, all edges on the middle path of the frame must be vertical, even those that do not have an arrowhead on them.[1] With this, the embedding of G implies an embedding of G_1 in $\pi \boxtimes P_\infty$ that is orientation-constrained, and we can hence extract a solution to the NAE-3SAT instance as Claim 13. This finishes the proof of Theorem 9.

C Row treedepth

Recall that S_∞ (the infinite *star*) is the tree that consists of one *center* that is adjacent to all other vertices (the *leaves*), with no restriction on its number of leaves.

Theorem 15. *It is NP-hard to test whether a tree is a subgraph of $S_\infty \boxtimes P_\infty$, even for a tree of pathwidth 2.*

Proof. We use a reduction from *3-partition*, where the input is a multi-set $A = \{a_1, \dots, a_{3n}\}$ of positive integers. The goal is to split these $3n$ integers into n groups that all sum to the same integer $B = \frac{1}{n}\sum_{i=1}^{3n} a_i$. This problem is strongly NP-hard [12], i.e., it remains NP-hard even if A is encoded in unary. We may assume that all input-numbers are multiples of 8 (otherwise multiply all of them by 8; this does not affect NP-hardness). We describe the construction of our tree T and at the same time also argue what any embedding Γ of T in $S_\infty \boxtimes P$ must look like. In $S_\infty \boxtimes P_\infty$, we call the P-extension of the center c the *center-row*; as in Observation 6 we use a degree-argument to force many vertices of T to be in the center-row, and finding enough space to hold all of them is the crucial idea for our reduction.

Tree T consists of a *frame* as well as a *paddle* for each a_i, $i = 1, \dots, 3n$, see also Fig. 7. The frame is a very long path, with most vertices on the path having 6 leaves attached. (These leaves are not shown in our picture.) The vertices with attached leaves are called *c-vertices* and are forced to be on the central row since all other vertices of $S_\infty \boxtimes P$ have degree 5. All other vertices of the frame are called *ℓ-vertices* because they could be on a *leaf-row* (the P-extension of a leaf of S_∞). The specific spacing along the path is as follows:

- Begin with $n(B{+}8)$ c-vertices (the *left blocker*). Since c-vertices must be on the central row, and no two central-row vertices are adjacent unless they are consecutive, this path (and similarly any path of c-vertices used below) occupies a consecutive set of vertices on the central row.
- Continue with B ℓ-vertices, followed by 8 c-vertices. The ℓ-vertices could be on a leaf-row, hence keep up to B vertices of the central row unused. We call this a *group-gap*.
- We create n consecutive group-gaps (we have $n = 2$ in Fig. 7).
- The last vertex Z of the last group-gap is called the *anchor*; the paddles (defined below) will attach at Z.
- Starting at Z, we alternate between three c-vertices and one ℓ-vertex that together define one *fold-gap* (it permits to omit one center-row vertex). There are $\frac{1}{8}n(B+8)$ fold-gaps.
- Finally we finish with $n(B{+}8)$ c-vertices (the *right blocker*).

Note that the left and right blocker are so long that no sub-path of ℓ-vertices could extend beyond them; in particular this forces all c-vertices that are not in the blockers to be between them in the central row.

Now for each $a_i \in A$, we define the *a_i-paddle*. This starts at anchor Z, continues with a path (the *handle*) that has $n(B{+}8){-}1$ ℓ-vertices, and culminates at the *blade*, which consists of a_i c-vertices. The handle is not long enough to extend beyond the blockers, so the c-vertices of the blade must be at $a_i \geq 2$ consecutive central-row vertices between the blockers. Since each fold-gap leaves at most one central-row vertex free, the blade must hence occupy central-row vertices left free by a group-gap. There are at most nB such central-row vertices in Γ, and they come in blocks of at most B consecutive central-row vertices each. By $\sum_{i=1}^{3n} a_i = nB$, it follows that in any realization Γ the group-gaps leave exactly n blocks of exactly B central-row vertices each, and the blades exactly fill these gaps, hence giving the desired partition of A.

We must still argue that if there is a solution to 3-partition, then we can embed T in $S_\infty \boxtimes P$, and for this, need the fold-gaps and $\Theta(n)$ leaves at the star. Embed first the frame as in Fig. 7, so all gaps leave the maximal possible number of central-row vertices free. (We also use 6 leaf-rows, not shown here, to embed the leaves attached at c-vertices.) We treat the center-row as if it were the x-axis with Z at the origin; this defines an x-coordinate $x(\cdot)$ for all embedded vertices with $x(Z) = 0$. Embed the blades of $a_1, \dots, a_{3n}$ in the group-gaps according to the solution to 3-partition. For $i = 1, \dots, 3n$, let v_i be the rightmost central-row vertex of the blade of a_i. To place the handle, we use two further leaf-rows, say ℓ'_i and ℓ''_i. We go from v_i diagonally rightward to ℓ'_i, then rightward for $|x(v_i)| - 1$ edges to reach x-coordinate -1. Hence we could now go to the anchor diagonally, but the handle is longer than this. Therefore we continue rightward for another $d_i := \frac{1}{2}(n(B{+}8) - |x(v_i)|)$ edges along ℓ'_i. Recall that each a_i (and hence also B) is divisible by 8. Since there are 8 c-vertices at each group-gap, and all group-gaps are completely filled by paddles, x-coordinate $x(v_i)$ is also divisible by 8. Thus d_i is divisible by 4, and the

[1]This argument would be simplified if we added arrow-heads everywhere, but then the graph would not be series-parallel.

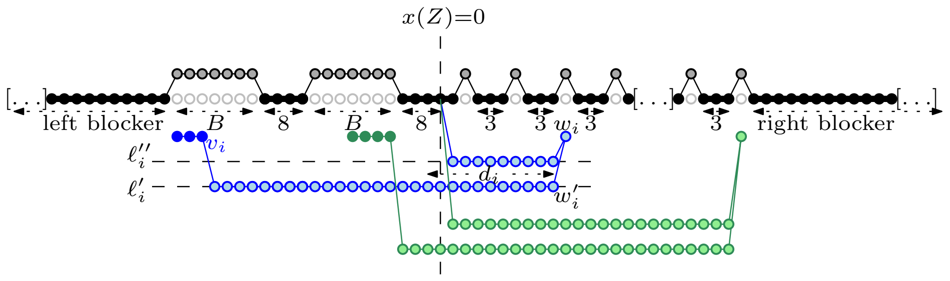


Figure 7: NP-hardness of embedding in $S_\infty \boxtimes P$, figure is not to scale. Filled dots represent c-vertices (hence have 6 leaves attached). We only show two paddles, one green and one blue.

vertex w_i' that we reach is one unit left of the central-row vertex w_i of some fold-gap. Go diagonally from w_i' to w_i, and from there diagonally back to $x(w_i')$ on the other leaf-row ℓ_i''. Then we go leftward along leaf-row ℓ_i'' to x-coordinate 1 and then diagonally to Z. In total we have used $|x(v_i)|-1+2d_i = n(B+8)-1$ vertices, which is exactly the length of the handle. Observe that vertex w_i cannot have been used by a different paddle (say the a_j-paddle) because $v_j \neq v_i$ are distinct central-row vertices, and their x-coordinates determine the fold-gap to be used.

Thus a solution to 3-partition gives an embedding of G in $S_\infty \boxtimes P$ and vice versa and the problem is NP-hard. Clearly we constructed a tree T; and if we removed the path π that defined the frame then all components of $T \setminus \pi$ are either singleton-vertices (at c-vertices of the frame) or caterpillars (at the paddles). Therefore T has pathwidth 2. $\square$

The same result also holds for the problem of embedding in $S_\infty \square P$. We use exactly almost the same tree T, except at each gap of the frame the path of ℓ-vertices is longer by two vertices and the handles have four more vertices. Details are left to the reader.

Our constructed trees have pathwidth 2. For a tree T of pathwidth 1, the answer to 'is $T \subset S_\infty \boxtimes P_\infty$' is trivial because the answer is always 'Yes': Such a tree is a subgraph of C_∞, and C_∞ can be embedded in $S_\infty \boxtimes P_\infty$ by placing the spine on the center-row.

On the Deque and Rique Numbers of Complete and Complete Bipartite Graphs

Michael A. Bekos* Michael Kaufmann† Maria Eleni Pavlidi‡ Xenia Rieger§

Abstract

Several types of linear layouts of graphs are obtained by leveraging known data structures; the most notable representatives are the stack and the queue layouts. In this content, given a data structure, one seeks to specify an order of the vertices of the graph and a partition of its edges into *pages*, such that the endpoints of the edges assigned to each page can be processed by the given data structure in the underlying order.

In this paper, we study deque and rique layouts of graphs obtained by leveraging the double-ended queue and the restricted-input double-ended queue (or deque and rique, for short), respectively. Hence, they generalize both the stack and the queue layouts. We focus on complete and complete bipartite graphs and present bounds on their deque- and rique-numbers, that is, on the minimum number of pages needed by any of these two types of linear layouts.

1 Introduction

Stack and queue layouts form two of the most studied types of linear layouts of graphs; they date back to 70's [9, 17] and over the years several remarkable results have been proposed in the literature [11, 16, 18, 19, 23]. For an introduction, refer to Section 2. Both layouts are defined by an underlying vertex order and an edge-partition into a certain number of so-called *pages* (*stacks* or *queues*, respectively), such that when restricting to a single page the endpoints of the edges assigned to it can be processed by the corresponding data structure in the order that appears in the underlying vertex order.

Since given a graph the natural goal is to find a layout of it that minimizes the number of used pages under the restrictions mentioned above, stack and queue layouts have naturally been leveraged to estimate the power of the respective data structures as a mean for representing graphs (for a wealth of other applications, e.g., to VLSI design and Graph Drawing, refer to [12]). The well-known *stack-number* (a.k.a. *book-thickness* or *page-number* in the literature) of a graph corresponds to the minimum number of stacks required by any of the stack layouts of it; the queue-number of a graph is defined symmetrically. In this context, it was recently shown that the stack-number of a graph cannot always be bounded by its corresponding queue-number [10], resolving a long-standing open question by Heath, Leighton and Rosenberg [16]; the other direction is still unknown.

A data structure that generalizes both the stack and the queue is the so-called *double-ended queue* or *deque*, for short.[1] As a matter of fact, the most common implementations of stacks and queues are derived by restricting corresponding implementations of deques. So, in this aspect, one naturally expects that the corresponding linear layouts that are obtained by employing the deque data structure for stipulating their edge-partitions will require fewer pages (called *deques* in this content) than those of stack or queue layouts, since, obviously, the latter form a special case of the former.

However, in contrast to the literature for stack and queue layouts, the corresponding literature for deque layouts is significantly reduced. To the best of our knowledge, there exists only one work introducing and studying deque layouts by Auer et al. [3], who provide a complete characterization of the graphs admitting 1-deque layouts (that is, deque layouts with a single deque): a graph admits a 1-deque layout if and only if it is a spanning subgraph of a planar graph with a Hamiltonian path; see also [2]. Even though the *deque-number* of a graph (that is, the minimum number of deques required by any of the deque layouts of the graph) has not been explicitly studied so far in the literature as a graph parameter, from the characterization by Auer et al. one can easily deduce the following.

Observation 1 (Auer et al. [3]) *The deque-number of a graph is at most half of its stack-number.*

Note that the queue-number is also a trivial upper bound on the deque-number of a graph. Observation 1,

*Department of Mathematics, University of Ioannina, Ioannina, Greece, bekos@uoi.gr

†Institute for Computer Science, University of Tübingen, Tübingen, Germany, michael.kaufmann@uni-tuebingen.de

‡Department of Mathematics, University of Ioannina, Ioannina, Greece, marialenaregie3@gmail.com

§Institute for Computer Science, University of Tübingen, Tübingen, Germany, xenia.rieger@student.uni-tuebingen.de

[1] While in a stack insertions and removals only occur at its head and in a queue insertions only occur at its head and removals only at its tail, a deque supports insertions and removals both at its head and its tail.

however, immediately implies improved upper bounds on the deque-number of several graph classes, e.g., the deque-number of the complete graph K_n is at most $\lceil \frac{n}{4} \rceil$ [9], the deque-number of the complete graph $K_{n,n}$ is at most $\lceil \frac{\lfloor 2n/3 \rfloor + 1}{2} \rceil$ [13], while the deque-number of treewidth-k graphs is at most $\lceil \frac{k+1}{2} \rceil$ [15]. Also, since there exist maximal planar graphs that do not have a Hamiltonian path (e.g., the n-vertex ones with an independent set of size greater than $\frac{n}{2}+2$), it follows by a well-known result by Yannakakis [23] that the deque-number of planar graphs is 2; see also [6, 24].

Another consequence of Observation 1 is that deque layouts cannot be characterized by means of forbidden patterns in the underlying linear order, as it is the case, e.g., for stack and queue layouts [17, 21]; the former do not allow two edges of the same page to cross (i.e., to have alternating endpoints), while in the latter no two edges of the same page nest (i.e., have nested endpoints). The reason for the lack of such a characterization for deque layouts is the fact that maximal planar graphs with a Hamiltonian path are the maximal graphs that admit 2-stack layouts and these layouts do not admit characterizations in terms of forbidden patterns in the underlying linear order [22]. A characterization in terms of forbidden patterns is possible, however, for a special type of deque layouts, which were recently introduced and are referred to as *rique layouts* [4], since the underlying data structure is of restricted input (so-called *restricted-input double-ended queue* or *rique*,[2] for short): a graph admits a 1-rique layout if and only if it admits a vertex order $\prec$ avoiding three edges (a, a'), (b, b') and (c, c') such that $a \prec b \prec c \prec b' \prec \{a', c'\}$.

Our contribution. In this work, we present bounds on the deque- and rique-numbers of complete and complete bipartite graphs. Especially, for deque layouts, the main research question that triggered our work is whether it is possible to obtain better bounds than the obvious ones that one can deduce from Observation 1 (or in other words, whether the deque data structure is more powerful for representing graphs than two stacks). Surprisingly enough, we prove that for the case of complete graphs, this is not the case (see Theorem 2), while for the case of complete bipartite graphs our upper bound shows that an improvement by a constant number is possible (to achieve this, however, we describe a rather complicated edge-to-deques assignment; see Theorem 5). For rique layouts, our contribution is twofold. First, we improve the upper bound on the rique-number of K_n from $\lceil \frac{n}{3} \rceil$ [4] to $\lfloor \frac{n-1}{3} \rfloor$ (see Theorem 4), which we prove to be tight up to $n = 30$ using an SAT-based approach (see Section 5). We complete our study with an upper bound of $\lfloor \frac{n-1}{2} \rfloor - 1$ on the rique-number of $K_{n,n}$.

[2]Formally, in a rique insertions occur only at the head, and removals occur both at the head and the tail. Thus, it is a special case of a deque and a generalization of a stack or of a queue.

2 Preliminaries

A *vertex order* $\prec$ of a graph G is a total order of its vertices, such that for any two vertices u and v of G, $u \prec v$ if and only if u precedes v in the order. We write $[u_1, \dots, u_k]$ if and only if $u_i \prec u_{i+1}$ for all $1 \leq i \leq k-1$. Let F be a set of $k \geq 2$ pairwise independent edges (u_i, v_i) of G, that is, $F = \{(u_i, v_i);\ i = 1, \dots, k\}$. If the order is $[u_1, \dots, u_k, v_k, \dots, v_1]$, then we say that the edges of F form a *k-rainbow*, while if the order is $[u_1, v_1, \dots, u_k, v_k]$, then the edges of F form a *k-necklace*. The edges of F form a *k-twist*, if the order is $[u_1, \dots, u_k, v_1, \dots, v_k]$; see Fig. 1. Two independent edges that form a 2-twist (2-rainbow, 2-necklace) are commonly referred to as *crossing* (*nested*, *disjoint*, respectively).

A *stack* is a set of pairwise non-crossing edges in $\prec$, while a *queue* is a set of pairwise non-nested edges in $\prec$. A *rique* is a set of edges in which no three edges (a, a'), (b, b') and (c, c') with $a \prec b \prec c \prec b' \prec \{a', c'\}$ exists in $\prec$. A deque is more difficult to describe due to the absence of a forbidden pattern. A relatively-simple way is the following. Assume that the vertices of a graph are arranged on a horizontal line ℓ from left to right according to $\prec$ (say, w.l.o.g., equidistantly). Then, each edge (v_i, v_j) with $v_i \prec v_j$ can be represented either (i) as a semi-circle that is completely above or completely below ℓ connecting u_i and u_j, or (ii) as two semi-circles on opposite sides of ℓ, one that starts at u_i and ends at a point p_{ij} of ℓ to the right of the last vertex of $\prec$ and one that starts at point p_{ij} and ends at u_j. With these in mind, a deque is a set of edges each of which can be represented with one of the two types (i) or (ii) that avoids crossings (such a representation is called *cylindric* in [3]); see Fig. 1d. A deque further allows classifying the edges into four categories: head-head, tail-tail, head-tail and tail-head; refer to the blue, light-blue, red and light-red edges of Fig. 1d, respectively. A *head-head* (*tail-tail*) edge is a type-(i) edge drawn above (below, respectively) ℓ. Symmetrically, a *head-tail* (*tail-head*) edge is a type-(ii) edge whose first part is above (below) ℓ, while its second part is below (above, respectively) ℓ. Given a deque layout L and a set of edges E, we write E_x to denote that all edges of E are of type-x in L, where $x \in \{hh, tt, ht, th\}$.

In view of the above definitions, a rique can be equivalently defined as a deque without tail-tail and tail-head edges [4]. Also, it is not difficult to see that the subset of the head-head or tail-tail edges of a deque induce a stack in $\prec$, while the set of the head-tail or tail-head edges of a deque induces a queue in $\prec$.

Since we focus on complete and complete bipartite graphs, for representing their linear layouts we use a

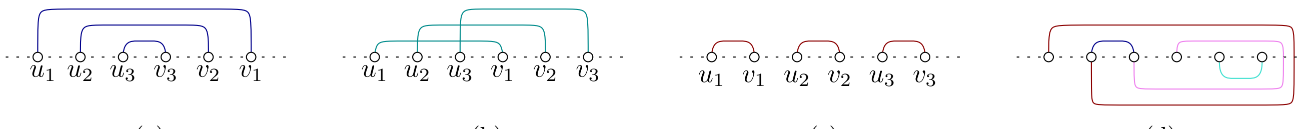

u_1 u_2 u_3 v_3 v_2 v_1
(a)
u_1 u_2 u_3 v_1 v_2 v_3
(b)
u_1 v_1 u_2 v_2 u_3 v_3
(c)
(d)

Figure 1: Illustration of: (a) a 3-rainbow, (b) a 3-twist, (c) a 3-necklace, and (d) a deque.

convenient way first introduced in [20] and subsequently used in several works [13, 14, 1]. Let $\prec$ be an order of the n vertices $v_1, \dots, v_n$ of a graph G such that $v_1 \prec \dots \prec v_n$. Then, each edge (v_i, v_j) of G with $i < j$ is mapped to point (i, j) of the $n \times n$ grid $H = [1, n] \times [1, n]$. A set of head-head or tail-tail edges of the same page (deque or rique) corresponds to a set of points on H whose union forms a monotonically decreasing curve on H [20]. A set of head-tail or tail-head edges of the same page (deque or rique) corresponds to a set of points on H whose union forms a monotonically increasing path on H [1]. If a deque contains head-tail and tail-head edges, special care is needed to avoid configurations not appearing in a cylindric layout.

3 Complete graphs

In this section, we study the deque- and rique-numbers of the complete graph K_n. As already mentioned, Observation 1 implies that $\lceil \frac{n}{4} \rceil$ is an easy-to-obtain upper bound on the deque-number of K_n. In the following, we prove that this bound is tight. To do so, we first give an estimation on the maximum number of edges that a graph admitting a k-deque layout can have.

Lemma 1 *A graph with n vertices admitting a deque layout with k pages has at most $(2k+1)n-5k-1$ edges.*

Proof. Let G be a graph with n vertices admitting a k-deque layout. Let also $v_1 \prec \dots \prec v_n$ be the linear order of the vertices of G. Since each deque induces a planar graph, it has at most $3n-6$ edges. However, the $n-1$ so-called *spine edges* (v_i, v_{i+1}), $i = 1, \dots, n-1$ can be added as head-head edges to every deque of the layout. So, every deque has at most $2n-5$ non-spine edges. Hence, in total G has $(2n-5)k+n-1$ edges. □

We are now ready to prove that the deque-number of the complete graph K_n is $\lceil \frac{n}{4} \rceil$.

Theorem 2 *The deque-number of K_n is $\lceil \frac{n}{4} \rceil$.*

Proof. The upper bound follows from Observation 1 and [9]. For the lower bound, let k be the number of deques of K_n. Since K_n has $\frac{n(n-1)}{2}$ edges, by Lemma 1, it follows that $(2k+1)n-5k-1 \geq \frac{n^2-n}{2}$, which implies:

$$k \geq \frac{n^2-3n+2}{4n-10} \quad \text{for } n \geq 3$$

In [7], we show that $\lceil \frac{n^2-3n+2}{4n-10} \rceil = \lceil \frac{n}{4} \rceil$, which completes the proof. □

For rique layouts, the analog of Lemma 1 is the following, which has been used to show a lower bound of $(1-\frac{\sqrt{2}}{2})(n-2)$ on the rique-number of K_n [4].

Lemma 3 (Bekos et al. [4]) *A graph with n vertices admitting a rique layout with k pages has at most $(2n+2)k-k^2+(n-3)$ edges.*

In the next theorem, we improve the best-known upper bound on the rique-number of K_n from $\lceil \frac{n}{3} \rceil$ [4] to $\lfloor \frac{n-1}{3} \rfloor$.

Theorem 4 *The rique-number of K_n is at most $\lfloor \frac{n-1}{3} \rfloor$.*

Proof. Assuming $n \bmod 3 = 0$, we prove that K_n admits a rique layout $\mathcal{L}$ with $\frac{n}{3}-1$ riques; the cases $n \bmod 3 \in \{1, 2\}$ are deferred to [7]. Our construction contains seven "special" pages, namely, the ones in $\{1, 2, 3, 4, \frac{n}{3}-3, \frac{n}{3}-2, \frac{n}{3}-1\}$; blue, red, green, dark-purple, gray, light-purple and yellow in Fig. 2a. The remaining pages of $\mathcal{L}$ are uniform.

Page 1 of $\mathcal{L}$ contains the following $2n$ edges:

- $\{(v_1, v_j), j = 2, \dots, n\}_{ht}$,
- $\{(v_i, v_n), i = 2, \dots, \frac{n}{3}\}_{ht}$,
- $\{(v_{\frac{n}{3}}, v_j), j = \frac{n}{3}+1, \dots, \frac{2n}{3}\}_{hh}$,
- $\{(v_{\frac{2n}{3}+1}, v_j), j = \frac{2n}{3}+2, \dots, n\}_{hh}$,
- $\{(v_{n-1}, v_n)\}_{hh}$.

Page 2 of $\mathcal{L}$ contains the following $2n-7$ edges:

- $\{(v_2, v_j), j = 3, \dots, n-1\}_{ht}$,
- $\{(v_i, v_{n-1}), i = 3, \dots, \frac{n}{3}+1\}_{ht}$,
- $\{(v_{\frac{n}{3}+1}, v_n)\}_{ht}$,
- $\{(v_{\frac{n}{3}+1}, v_j), j = \frac{n}{3}+2, \dots, \frac{2n}{3}\}_{hh}$,
- $\{(v_{\frac{2n}{3}}, v_j), j = \frac{2n}{3}+1, \dots, n\}_{hh}$.

Page 3 of $\mathcal{L}$ contains the following $2n-5$ edges:

- $\{(v_3, v_j), j = 4, \dots, n-2\}_{ht}$,
- $\{(v_i, v_{n-2}), i = 4, \dots, \frac{n}{3}+1\}_{ht}$,
- $\{(v_{\frac{2n}{3}+2}, v_j), j = n-2, \dots, n\}_{ht}$,
- $\{(v_{\frac{n}{3}+1}, v_{\frac{2n}{3}+1})\}_{hh}$,

- $\{(v_{\frac{n}{3}+2}, v_j), j = \frac{2n}{3} - 1, \frac{2n}{3}, \frac{2n}{3} + 1\}_{hh}$,
- $\{(v_{\frac{n}{3}+3}, v_j), j = \frac{n}{3} + 4, \dots, \frac{2n}{3} - 1\}_{hh}$,
- $\{(v_{\frac{2n}{3}+2}, v_j), j = \frac{2n}{3} + 3, \dots, n - 3\}_{hh}$,
- $\{(v_{n-3}, v_j), j = n - 2, n - 1, n\}_{hh}$.

For $p = 4, \dots, \frac{n}{3} - 4$, page p of $\mathcal{L}$ contains the following $\frac{n}{3} - 2p + 3$ edges:

- $\{(v_p, v_j), j = p + 1, \dots, n - p + 1\}_{ht}$,
- $\{(v_i, v_j), i = p + 1, \dots \frac{n}{3} + 1, j = n - p + 1\}_{ht}$,
- $\{(v_i, v_j), i = \frac{n}{3} + (p + 1), j = n - p + 1, \dots, n\}_{ht}$,
- $\{(v_i, v_j), i = \frac{n}{3} + (p+1), j = \frac{2n}{3} + (p-2), \dots, n-p\}_{hh}$,
- $\{(v_i, v_j), i = n - p + 1, j = n - p, \dots, n\}_{hh}$,
- $\{(v_i, v_j), i = \frac{n}{3} + (p+2), j = \frac{n}{3} + (p+3), \dots, \frac{2n}{3} + (p-2)\}_{hh}$.

Page $\frac{n}{3} - 3$ of $\mathcal{L}$ contains the following $\frac{4n}{3} + 6$ edges:

- $\{(v_{\frac{n}{3}-3}, v_j), j = \frac{n}{3} - 2, \dots, \frac{2n}{3} + 4\}_{ht}$,
- $\{(v_i, v_{\frac{2n}{3}+4}), i = \frac{n}{3} - 2, \dots, \frac{n}{3} + 1\}_{ht}$,
- $\{(v_{\frac{n}{3}+3}, v_j), j = \frac{2n}{3} + 4, \dots, n - 1\}_{ht}$,
- $\{(v_{\frac{2n}{3}+3}, v_j), j = n - 1, n\}_{ht}$,
- $\{(v_{\frac{n}{3}+3}, v_j), j = \frac{2n}{3}, \dots, \frac{2n}{3} + 3\}_{hh}$,
- $\{(v_{\frac{n}{3}+4}, v_j), j = \frac{n}{3} + 5, \dots, \frac{2n}{3}\}_{hh}$,
- $\{(v_{\frac{2n}{3}+3}, v_j), j = \frac{2n}{3} + 3, \dots, n - 2\}_{hh}$,
- $\{(v_{n-2}, v_j), j = n - 1, n\}_{hh}$.

Page $\frac{n}{3} - 2$ of $\mathcal{L}$ contains the following $\frac{4n}{3} + 3$ edges:

- $\{(v_{\frac{n}{3}-2}, v_j), j = \frac{n}{3} - 1, \dots, \frac{2n}{3} + 3\}_{ht}$,
- $\{(v_i, v_{\frac{2n}{3}+3}), i = \frac{n}{3} - 1, \dots, \frac{n}{3} + 1\}_{ht}$,
- $\{(v_{\frac{n}{3}+2}, v_j), j = \frac{2n}{3} + 3, \dots, n\}_{ht}$,
- $\{(v_{\frac{n}{3}+3}, v_n)_{ht}$,
- $\{(v_{\frac{n}{3}+4}, v_j), j = \frac{2n}{3} + 1, \dots, n\}_{hh}$,
- $\{(v_{\frac{n}{3}+5}, v_j), j = \frac{n}{3} + 6, \dots, \frac{2n}{3} + 1\}_{hh}$.

Page $\frac{n}{3} - 1$ of $\mathcal{L}$ contains the following $n + 9$ edges:

- $\{(v_{\frac{n}{3}-1}, v_j), j = \frac{n}{3}, \dots, \frac{2n}{3} + 2\}_{ht}$,
- $\{(v_i, v_{\frac{2n}{3}+2}), i = \frac{n}{3}, \dots, \frac{n}{3} + 2\}_{ht}$,
- $\{(v_{\frac{2n}{3}-2}, v_j), j = n - 5, \dots, n\}_{ht}$,
- $\{(v_{\frac{n}{3}+2}, v_j), j = \frac{n}{3} + 3, \dots, \frac{2n}{3} - 2\}_{hh}$,
- $\{(v_{\frac{2n}{3}-1}, v_j), j = \frac{2n}{3}, \dots, n\}_{hh}$.

So, in total $\mathcal{L}$ has $(2n-1) + (2n-6) + (2n-4) + (\frac{5n}{3} - 5) + \sum_{p=5}^{\frac{n}{3}-4} (\frac{5n}{3} - p + 1) + (n + 14) + (\frac{4n}{3} + 3) + (\frac{4n}{3} + 3) = \frac{n(n-1)}{2}$ edges. Since no two edges have been assigned to the same rique and all edges in the same rique form a cylindric layout, it follows that the rique number of K_n is at most $\lfloor \frac{n-1}{3} \rfloor$ when $n \bmod 3 = 0$. □

Remark 1 Using the SAT formulation that we present in Section 5 we were able to show that the upper bound of Theorem 4 is tight for all values of $n \leq 30$. However, we were not able to show a matching lower bound. In view of these observations, we conjecture in Section 6 that the rique-number of K_n is exactly $\lfloor \frac{n-1}{3} \rfloor$.

4 Complete bipartite graphs

In this section, we study the deque- and rique-numbers of the complete bipartite graph $K_{n,n}$. Let the two parts of $K_{n,n}$ be $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ with $|A| = |B| = n$. W.l.o.g., we may assume that in the computed layouts $a_1 \prec \dots \prec a_n$ and $b_1 \prec \dots \prec b_n$ holds.

Theorem 5 *The deque-number of $K_{n,n}$ is at most $\lceil \frac{n}{3} \rceil$.*

Proof. Assume that $n \bmod 3 = 0$; the remaining cases follow from this one. We describe a deque layout $\mathcal{L}$ of $K_{n,n}$ with $\frac{n}{3}$ deques, in which the underlying order is: $a_1 \prec \dots \prec a_{n/3} \prec b_1 \dots \prec b_{2n/3} \prec a_{n/3+1} \prec \dots \prec a_n \prec b_{2n/3+1} \prec \dots \prec b_n$.

Part 1: We start by describing how the edges between $a_{n/3+1}, \dots, a_n$ and $b_1, \dots, b_{2n/3}$ are assigned to the pages of $\mathcal{L}$; see Fig. 2b.

Page $\frac{n}{3}$ of $\mathcal{L}$ contains the following $\frac{4n}{3} - 1$ edges:

- $\{(a_{\frac{n}{3}+1}, b_j), j = 1, 2, 3\}_{ht}$,
- $\{(a_i, b_3), i = \frac{n}{3} + 2, \dots, n\}_{ht}$,
- $\{(a_n, b_j), j = 3, \dots, \frac{2n}{3}\}_{ht}$.

Page $\frac{n}{3} - 1$ of $\mathcal{L}$ contains the following $n + 2$ edges:

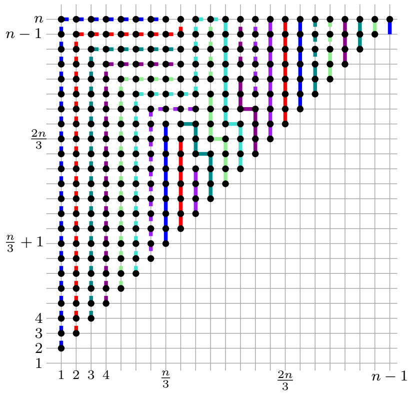
- $\{(a_{n-1}, b_j), j = \frac{2n}{3} - 2, \dots, \frac{2n}{3}\}_{ht}$,
- $\{(a_{n-1}, b_1), (a_{n-2}, b_1), (a_{n-2}, b_2), (a_{n-3}, b_2)\}_{tt}$,
- $\{(a_i, b_j), (a_{i-1}, b_j), i = n-3, \dots, \frac{2n}{3} + 2, j = 4, \dots, \frac{n}{3} - 2\}_{tt}$,
- $\{(a_i, b_{\frac{n}{3}-1}, i = \frac{2n}{3} + 2, \dots, \frac{2n}{3}\}_{tt}$,
- $\{(a_{\frac{2n}{3}}, b_j), j = \frac{n}{3}, \frac{n}{3} + 1\}_{tt}$,
- $\{(a_i, b_{\frac{n}{3}+2}), i = \frac{n}{3} + 1, \dots, \frac{2n}{3}\}_{tt}$.

Page $\frac{n}{3} - 2$ of $\mathcal{L}$ contains the following $\frac{2n}{3} + 9$ edges:

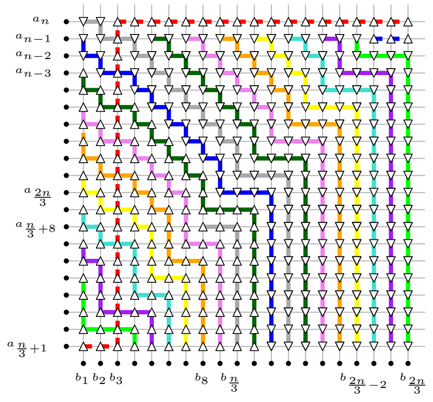
- $\{(a_i, b_1), i = \frac{n}{3} + 2, \dots, \frac{n}{3} + 5\}_{hh}$,
- $\{(a_{\frac{n}{3}+2}, b_2), (a_{\frac{n}{3}+2}, b_4), (a_{\frac{n}{3}+1}, b_4)\}_{hh}$,
- $\{(a_{n-1}, b_{\frac{2n}{3}-3})\}_{tt}$,
- $\{(a_{n-2}, b_j, j = \frac{2n}{3} - 3, \dots, \frac{2n}{3}\}_{tt}$,
- $\{(a_i, b_{\frac{2n}{3}}), i = \frac{n}{3} + 1, \dots, n - 3\}_{tt}$.

Page $\frac{n}{3} - 3$ of $\mathcal{L}$ contains the following $\frac{2n}{3} + 13$ edges:

- $\{(a_i, b_1), i = \frac{n}{3} + 6, \frac{n}{3} + 7\}_{hh}$,



(a)



(b)

Figure 2: The grid representation of (a) a rique layout of K_n with $n \bmod 3 = 0$ (grid points covered by a solid (dashed) path correspond to head-head (head-tail, resp.) edges), (b) Part 1 of Theorem 5 (a grid point represented by a triangle pointing up (down) corresponds to a head-$\star$ (tail-$\star$, resp.) edge; grid points covered by a solid line are head-head or tail-tail edges; the dashed covered ones are head-tail or tail-head).

- $\{(a_i, b_2), i = \frac{n}{3}+3, \ldots, \frac{n}{3}+6\}_{hh}$,
- $\{(a_{\frac{n}{3}+3}, b_4)\}_{hh}$,
- $\{(a_i, b_5), i = \frac{n}{3}+1, \ldots, \frac{n}{3}+3\}_{hh}$,
- $\{(a_{n-1}, b_j), j = \frac{2n}{3}-5, \frac{2n}{3}-4\}_{tt}$,
- $\{(a_i, b_{\frac{2n}{3}-4}), i = n-2, n-3\}_{tt}$
- $\{(a_{n-3}, b_j), j = \frac{2n}{3}-3, \frac{2n}{3}-2\}_{tt}$,
- $\{(a_i, b_{\frac{2n}{3}-1}), i = \frac{n}{3}+1, \ldots, n-3\}_{tt}$.

Page 1 of $\mathcal{L}$ contains the following $n+11$ edges:

- $\{(a_{\frac{n}{3}+8}, b_j), j = 8, 9, 10\}_{hh}$,
- $\{(a_i, b_{10}), i = \frac{n}{3}+1, \ldots, \frac{n}{3}+7\}_{hh}$,
- $\{(a_n, b_1), (a_n, b_2), (a_{n-1}, b_2)\}_{tt}$,
- $\{(a_i, b_j), (a_{i-1}, b_j), i = n-1, \ldots, \frac{2n}{3}+3, j = 4, \ldots, \frac{n}{3}\}_{tt}$,
- $\{(a_i, b_{\frac{n}{3}}), i = \frac{2n}{3}+1, \ldots, \frac{2n}{3}+3\}_{tt}$,
- $\{(a_{\frac{2n}{3}+1}, b_j), j = \frac{n}{3}+1, \ldots, \frac{n}{3}+3\}_{tt}$,
- $\{(a_i, b_{\frac{n}{3}+3}), i = \frac{n}{3}+1, \ldots, \frac{2n}{3}\}_{tt}$.

For $p = \frac{n}{3}-7, \ldots, \frac{n}{3}-4$, page p of $\mathcal{L}$ contains $2n-4p+1$ edges:

- $\{(a_i, b_{\frac{n}{3}-p+2}), i = \frac{n}{3}+1, \ldots, \frac{2n}{3}-p\}_{hh}$,
- $\{(a_{\frac{2n}{3}-p}, b_j), j = \frac{n}{3}-p, \frac{n}{3}-p+1\}_{hh}$,
- $\{(a_i, b_{\frac{n}{3}-p}), i = \frac{2n}{3}-p+1, \ldots, \frac{2n}{3}-p+3\}_{hh}$,
- $\{(a_i, b_1), i = n-2p+1, n-2p\}_{hh}$,
- $\{(a_i, b_2), i = n-2p, n-2p-1\}_{hh}$,
- $\{(a_i, b_j), (a_{i-1}, b_j), i = n-2p-1, \ldots, n-2p+3, j = 4, \ldots, 6\}_{hh}$,
- $\{(a_i, b_{\frac{n}{3}+p+2}), i = \frac{n}{3}+1, \ldots, \frac{2n}{3}+p\}_{tt}$,
- $\{(a_{\frac{2n}{3}+p}, b_j), j = \frac{n}{3}+p-1, \ldots, \frac{n}{3}+p+1\}_{tt}$,
- $\{(a_{\frac{2n}{3}+p+1}, b_{\frac{n}{3}+p-1})\}_{tt}$,
- $\{(a_i, b_j), (a_i, b_{j+1}), i = n-1, \ldots, n-5, j =, \ldots, \frac{n}{3}+p-2\}_{tt}$.

For $p = 2, \ldots, \frac{n}{3}-8$, page p of $\mathcal{L}$ contains $2n-4p+2$ edges:

- $\{(a_i, b_1), i = n-2p+1, n-2p\}_{hh}$,
- $\{(a_i, b_2), i = n-2p, n-2p-1\}_{hh}$,
- $\{(a_i, b_j), (a_{i-1}, b_j) i = n-2p-1, \ldots, \frac{2n}{3}+4-p, j = 4, \ldots, \frac{n}{3}-p-1\}_{hh}$,
- $\{(a_i, b_{\frac{n}{3}-p}), i = \frac{2n}{3}-p+3, \ldots, \frac{2n}{3}-p+1\}_{hh}$,
- $\{(a_{\frac{2n}{3}-p}, b_j), j = \frac{n}{3}-p+1, \frac{n}{3}-p+2\}_{hh}$,
- $\{(a_i, b_{\frac{n}{3}-p+3}), i = \frac{n}{3}+1, \ldots, \frac{2n}{3}-p+1\}_{hh}$,
- $\{(a_i, b_j), (a_i, b_{j+1}), i = n-1, \ldots, \frac{2n}{3}+p+2, j = 5, \ldots, \frac{n}{3}+p-2\}_{tt}$,
- $\{(a_i, b_{\frac{n}{3}+p-1}), i = \frac{2n}{3}+p, \frac{2n}{3}+p+1\}_{tt}$,
- $\{(a_{\frac{2n}{3}+p}, b_j), i = \frac{n}{3}+p, \frac{n}{3}+p+1\}_{tt}$,
- $\{(a_i, b_{\frac{n}{3}+p+2}), i = \frac{n}{3}+1, \ldots, \frac{2n}{3}+p\}_{tt}$.

Parts 2,3 and 4 (with $\frac{2n^2}{9}$, $\frac{n^2}{9}$ and $\frac{2n^2}{9}$ edges, resp.) and the correctness proof are discussed in [7]. □

Due to space constraints the proof of the following theorem is completely deferred to [7].

Theorem 6 *The rique-number of the complete bipartite graph $K_{n,n}$ is at most $\lfloor \frac{n-1}{2} \rfloor - 1$.*

5 SAT formulation

In this section, we present a SAT formulation for the problem of testing whether a given graph with n vertices and m edges admits a deque layout with p of deques; an implementation has already been incorporated in [5], whose source code is available at `https://github.com/linear-layouts/SAT`. However, before describing our formulation, we deem important to state that, in [4], Bekos et al. have already presented a corresponding SAT formulation, when the p pages are riques. However, their approach heavily relies on the fact that this specific type of linear layouts can be characterized by means of a forbidden pattern in the underlying order (similar to the corresponding ones for stack and queue layouts [8]). Given that dequeue layouts cannot be characterized by means of such forbidden patterns in the underlying order, we need a slightly different approach.

Similar to [4], our approach is an extension of the one in [8] for the stack layout problem, in which there exist three different types of variables, denoted by σ, ϕ, and χ, with the following meanings: (i) for a pair of vertices u and v, variable $\sigma(u,v)$ is `true`, if and only if u is to the left of v along the spine, (ii) for an edge e and a page i, variable $\phi_i(e)$ is `true`, if and only if edge e is assigned to page i of the book, and (iii) for a pair of edges e and e', variable $\chi(e,e')$ is `true`, if and only if e and e' are assigned to the same page. Hence, there exist in total $O(n^2+m^2+pm)$ variables, while a set of $O(n^3)$ clauses ensures that the underlying order is indeed linear; for details see [8]. To overcome the issue that arises in the absence of forbidden pattern, we introduce $4pm$ variables, such that variable $\tau_i(e,x)$ with $x \in \{hh, ht, th, tt\}$ is `true`, if and only if the type of edge e at page i is x. We ensure that each edge has at least one of the allowed types, by introducing the following clause for each edge e: $\bigvee_{i=1}^{p}(\tau_i(e,hh) \vee \tau_i(e,ht) \vee \tau_i(e,th) \vee \tau_i(e,tt))$. With these variables, we can express different configurations that cannot occur in a deque layout as clauses in the SAT formula. These clauses are obtained by avoiding crossings between all edge types in the cylindric representation of the graph. E.g., to express the different configurations that cannot occur for a head-head edge $e=(u,v)$ and a head-tail edge $e'=(u',v')$, we introduce the following clause for each page i of the layout[3]:

$$
\begin{aligned}
&\phi_i(e) \wedge \phi_i(e') \wedge \tau_i(e,hh) \wedge \tau_i(e',ht) \rightarrow \\
&\neg(\sigma(u,u') \wedge \sigma(u',v) \wedge \sigma(v,v')) \quad u \neq v \neq u' \\
&\wedge \neg(\sigma(v,u') \wedge \sigma(u',u) \wedge \sigma(u,v')) \quad u \neq v \neq u' \\
&\wedge \neg(\sigma(u,v') \wedge \sigma(v',v) \wedge \sigma(v,u')) \quad u \neq v \neq v' \\
&\wedge \neg(\sigma(v,v') \wedge \sigma(v',u) \wedge \sigma(u,u')) \quad u \neq v \neq v' \\
&\wedge \neg(\sigma(u,u') \wedge \sigma(u',v') \wedge \sigma(v',v)) \quad u \neq u' \\
&\wedge \neg(\sigma(u,v') \wedge \sigma(v',u') \wedge \sigma(u',v)) \quad u \neq v' \\
&\wedge \neg(\sigma(v,u') \wedge \sigma(u',v') \wedge \sigma(v',u)) \quad v \neq u' \\
&\wedge \neg(\sigma(v,v') \wedge \sigma(v',u') \wedge \sigma(u',u)) \quad v \neq v'
\end{aligned}
$$

We introduce a clause similar to the one above for each pair of types of edges, yielding in total $O(pm^2)$ clauses. This completes the construction of the formula. Note that the formulation can be easily adjusted for rique layouts by introducing for each edge e and each page i the following clause forbidding tail-head and tail-tail edges: $\neg\tau_i(e,th) \wedge \neg\tau_i(e,tt)$. Somehow unexpectedly, this simple adjustment was more efficient in practice than the one by Bekos et al. [4], which is based on implementing the forbidden pattern of rique layouts.

Findings. The implementation was extremely helpful, in general, for developing all upper bounds of this paper. It further shows that the upper bound of Theorem 4 is tight for all values of $n \leq 30$ (see Remark 1). Another notable observation is that for $K_{n,n}$ it is possible to obtain a better upper bound than the one of Observation 1 (or in other words that k deques are strictly more powerful than $2k$ stacks): Our implementation shows that $K_{3n,3n}$ with $n \in \{2,3,4,5\}$ needs $n+1$ stacks, while the solver provided solutions with n deques for the corresponding values of n. Note that this result would be implied (for any n) by Theorem 5, if the bound [13] on the stack number of $K_{n,n}$ was shown to be tight.

6 Open Problems

We conclude with some open problems: (i) We conjecture that the bound of Theorem 4 is tight. (ii) As mentioned in the introduction, the deque-number of planar graphs is 2. We conjecture that also their rique-number is 2. (iii) Another natural direction to follow is to extend the study to other classes of graphs, as it is the case with the corresponding stack- and queue-numbers. (iv) Studying inclusion relationships is also of interest, e.g., the class of graphs admitting 1-deques is not a subclass of the class of graphs admitting 1-rique, 1-stack layouts, as a maximal planar graph with a Hamiltonian path plus an edge belongs to the former but not to the latter. What about the other direction? (v) Related to our research is also the problem of closing the gap between the lower bound of $\lceil \frac{n}{2} \rceil$ and the upper bound of $\lfloor \frac{2n}{3} \rfloor + 1$ [13] on the stack number of $K_{n,n}$.

[3]Note that some parts of the clause appear only certain conditions apply on the endpoints of e and e'. These conditions are listed next to the corresponding parts, such that if a condition is not fulfilled, then the corresponding part has to be omitted.

References

[1] J. M. Alam, M. A. Bekos, M. Gronemann, M. Kaufmann, and S. Pupyrev. The mixed page number of graphs. *Theor. Comput. Sci.*, 931:131–141, 2022.

[2] C. Auer. *Planar graphs and their duals on cylinder surfaces*. PhD thesis, Universität Passau, 2014.

[3] C. Auer, C. Bachmaier, F. Brandenburg, W. Brunner, and A. Gleißner. Plane drawings of queue and deque graphs. In U. Brandes and S. Cornelsen, editors, *Graph Drawing*, volume 6502 of *LNCS*, pages 68–79. Springer, 2010.

[4] M. A. Bekos, S. Felsner, P. Kindermann, S. G. Kobourov, J. Kratochvíl, and I. Rutter. The rique-number of graphs. In P. Angelini and R. von Hanxleden, editors, *Graph Drawing and Network Visualization*, volume 13764 of *LNCS*, pages 371–386. Springer, 2022.

[5] M. A. Bekos, M. Haug, M. Kaufmann, and J. Männecke. An online framework to interact and efficiently compute linear layouts of graphs. *CoRR*, abs/2003.09642, 2020.

[6] M. A. Bekos, M. Kaufmann, F. Klute, S. Pupyrev, C. N. Raftopoulou, and T. Ueckerdt. Four pages are indeed necessary for planar graphs. *J. Comput. Geom.*, 11(1):332–353, 2020.

[7] M. A. Bekos, M. Kaufmann, M. E. Pavlidi, and X. Rieger. On the deque and rique numbers of complete and complete bipartite graphs. *CoRR*, abs/2306.15395, 2023.

[8] M. A. Bekos, M. Kaufmann, and C. Zielke. The book embedding problem from a sat-solving perspective. In E. Di Giacomo and A. Lubiw, editors, *Graph Drawing and Network Visualization*, volume 9411 of *LNCS*, pages 125–138. Springer, 2015.

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

[10] V. Dujmovic, D. Eppstein, R. Hickingbotham, P. Morin, and D. R. Wood. Stack-number is not bounded by queue-number. *Comb.*, 42(2):151–164, 2022.

[11] V. Dujmovic, G. Joret, P. Micek, P. Morin, T. Ueckerdt, and D. R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020.

[12] V. Dujmović and D. R. Wood. On linear layouts of graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004.

[13] H. Enomoto, T. Nakamigawa, and K. Ota. On the pagenumber of complete bipartite graphs. *J. Comb. Theory, Ser. B*, 71(1):111–120, 1997.

[14] S. Felsner, L. Merker, T. Ueckerdt, and P. Valtr. Linear layouts of complete graphs. In H. C. Purchase and I. Rutter, editors, *Graph Drawing and Network Visualization*, volume 12868 of *LNCS*, pages 257–270. Springer, 2021.

[15] J. L. Ganley and L. S. Heath. The pagenumber of k-trees is $O(k)$. *Discrete Applied Mathematics*, 109(3):215–221, 2001.

[16] L. S. Heath, F. T. Leighton, and A. L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discrete Math.*, 5(3):398–412, 1992.

[17] L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–958, 1992.

[18] M. Hoffmann and B. Klemz. Triconnected planar graphs of maximum degree five are subhamiltonian. In M. A. Bender, O. Svensson, and G. Herman, editors, *ESA*, volume 144 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[19] P. Jungeblut, L. Merker, and T. Ueckerdt. A sublinear bound on the page number of upward planar graphs. In J. S. Naor and N. Buchbinder, editors, *ACM-SIAM SODA*, pages 963–978. SIAM, 2022.

[20] D. J. Muder, M. L. Weaver, and D. B. West. Pagenumber of complete bipartite graphs. *J. Graph Theory*, 12(4):469–489, 1988.

[21] T. Ollmann. On the book thicknesses of various graphs. In F. Hoffman, R. Levow, and R. Thomas, editors, *Southeastern Conference on Combinatorics, Graph Theory and Computing*, volume VIII of *Congressus Numerantium*, page 459, 1973.

[22] A. Wigderson. The complexity of the Hamiltonian circuit problem for maximal planar graphs. Technical Report TR-298, EECS Department, Princeton University, 1982.

[23] M. Yannakakis. Embedding planar graphs in four pages. *J. Comput. Syst. Sci.*, 38(1):36–67, 1989.

[24] M. Yannakakis. Planar graphs that need four pages. *J. Comb. Theory, Ser. B*, 145:241–263, 2020.

Dynamic Schnyder woods

Sujoy Bhore[*] Prosenjit Bose[†] Pilar Cano[‡] Jean Cardinal[§] John Iacono[¶]

Abstract

A *Schnyder wood* of a triangulation is a partition of its interior edges into three oriented rooted trees (i.e., a three-colored and oriented triangulation). A *flip* in a Schnyder wood is a local operation that transforms one Schnyder wood into another, possibly of another triangulation. Two types of flips in a Schnyder wood have been introduced: *colored flips*, that change the underlying triangulation, and *cycle flips*, that transform a Schnyder wood into another Schnyder wood of the same triangulation. A *flip graph* is defined for each of these two types of flips. In this paper, we study the relationship between these two types of flips and their corresponding flip graphs. We show that a cycle flip can be obtained from linearly many colored flips. We also give an explicit upper bound of $O(n^2)$ on the diameter of the colored flip graph. Moreover, a data structure is given to dynamically maintain a Schnyder wood over a sequence of colored flips which supports queries in $O(\log n)$ time per flip or query.

1 Introduction

Schnyder in his seminal work proved that every planar graph with $n \geq 3$ vertices has a plane straight-line drawing in an $(n-2) \times (n-2)$ grid [29, 30]. This result was achieved in two parts: First, it was shown that every triangulation (a maximal planar graph) admits a decomposition of its interior edges into three trees, called *Schnyder wood*; Then, by using the Schnyder wood, a straight line embedding can be achieved. The Schnyder tree partitions are an important concept in the area of graph drawing and order dimension theory; see [3, 16, 17, 19, 20]. Schnyder woods have been widely used to obtain combinatorial and algorithmic results for a wide range of problems from various domains, e.g., geometric spanners [5], optimal encoding and uniform sampling of planar maps [26], compact data structures [12], grid drawing [3, 22, 30], etc. Moreover, the connection between Schnyder wood and orthogonal surfaces has been explored over the years; see [4, 18, 21, 22]. Recently, Castelli Aleardi [11] considered *balanced Schnyder woods*, in which the number of incoming edges of each color at each vertex is balanced, and provided linear time heuristics.

An *edge flip* in a triangulation T is a local operation that transforms T into another triangulation T′ that differs by exactly one edge and two face triangles. This operation leads to the definition of a *flip graph* where each triangulation with n vertices represents a vertex and two triangulations are adjacent if they differ by exactly one flip. This graph has been widely studied [7, 8, 27, 34]. In particular, the diameter of the edge flip graph restricted to triangulations embedded in the plane is $\Theta(n)$ [28, 32].

A *flip* in a Schnyder wood is a local operation that transforms one Schnyder wood into another. Two types of flips in a Schnyder wood have been introduced: *colored flip* and *cycle flip* (see Section 3). A corresponding *flip graph* is defined for each of these two types of flips, the vertex sets of which are the Schnyder woods, and two Schnyder woods are adjacent if they can be transformed into each other by one flip. Brehm [9] and Bonichon [6] showed that the cycle flip graph and the *colored* flip graph is connected, respectively. As for the diameter, it is known that the cycle flip distance between two Schnyder woods is $O(n^2)$ [1, 9, 33]. Tetali et al. [33] considered both cycle and *colored* flips in natural Markov chains for sampling uniformly from the set of 3 orientations. In [33] it is also shown implicitly that the *colored* flip graph has $O(n^3)$ diameter.

Dynamically maintaining a Schnyder wood of a planar graph is motivated by the existence of efficient algorithms for testing the planarity of a fully-dynamic graph [15, 24]. Recently, Christiansen et al. [13] considered the dynamic edge orientation problem, where the goal is to orient edges in a way that bounds the maximum out-degree as the graph is subject to insertions and deletions of edges. Moreover, they noted that their result on dynamic planar graphs implies that, it is not possible to maintain a dynamic Schnyder Wood explicitly of a graph with sublinear amortized update time.

Our Contribution. We describe Schnyder woods and related constructions in Section 2. In Section 3 we show that if an edge e admits a diagonal flip in a triangulation T, then there exists a Schnyder wood R of T where the oriented edge e admits a colored flip in R (Section 3.1). Later, we show that a cycle flip can be obtained from linearly many colored flips (Section 3.2).

[*]IIT Bombay, India, sujoy@cse.iitb.ac.in
[†]Carleton University, Canada, jit@scs.carleton.ca
[‡]mpilarcanovi@gmail.com
[§]ULB, Belgium, jean.cardinal@ulb.be
[¶]ULB, Belgium, jiacono@ulb.be

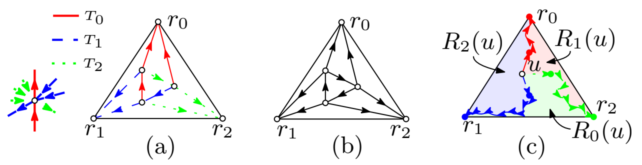


Figure 1: (a) Example of the counter-clockwise order of the edges entering and leaving the vertex (*left*) and a Schnyder wood (*right*). (b) A 3-orientation. (c) Each colored region with its respectively colored path represents a region $R_i(u)$.

Using these two results, we prove an upper bound of $O(n^2)$ on the diameter of the flip graph of Schnyder woods defined by colored flips. Finally, in Section 4 we present a data structure to dynamically maintain a Schnyder wood implicitly under colored flips while supporting queries to a corresponding straight line embedding over a sequence of colored flips in $O(\log n)$ amortized time per update or query. Omitted prrofs can be found in the appendix and in [2].

2 Schnyder Woods

In this section, we define a Schnyder wood and two other structures that are a bijection with Schnyder wood.

A *triangulation* is a maximal planar graph (all faces are triangles) with a fixed outer face. A *Schnyder wood* of a triangulation $\mathtt{T}$ is a partition of its interior edges into three sets T_0, T_1 and T_2 of directed edges such that for each interior vertex u the following holds:

1. Vertex u has out-degree exactly one in each of T_0, T_1 and T_2 in counter-clockwise order.
2. All incoming edges of T_i adjacent to u occur between the outgoing edge of T_j and T_k for distinct $i, j, k \bmod 3$. See Fig. 1(a).

Each tree T_i of a Schnyder wood has as root r_i, one of the vertices in its outer face, and each vertex in the outer face is a sink in the directed graph defined by the given Schnyder wood. Note that a Schnyder wood can be represented as a 3 coloring of its interior edges. See Fig. 1(a). Schnyder defined Schnyder wood of triangulations in [29,30] and proved that any triangulation with $n \geq 3$ vertices has a Schnyder wood.

A *3-orientation* of a triangulation $\mathtt{T} = (V \cup \{r_0, r_1, r_2\}, E)$ is an orientation of the edges of $\mathtt{T}$ such that each vertex has out-degree 3 except three special vertices r_0, r_1, r_2 that are sinks and define the outer face of $\mathtt{T}$. See Fig. 1(b)

In [14] de Fraysseix and Ossona de Mendez showed that any triangulation $\mathtt{T}$ admits a 3-orientation of its interior edges and that the Schnyder woods of a triangulation $\mathtt{T}$ form a bijection with 3-orientations of $\mathtt{T}$.

A *barycentric representation*[1] of a triangulation $\mathtt{T}$ is an injective function $u \in V(\mathtt{T}) \to (u_0, u_1, u_2) \in \mathbb{R}^3$ that satisfies the conditions:

1. $u_0 + u_1 + u_3 = 1$ for all vertices $u \in V(\mathtt{T})$.
2. For each edge uv and each vertex $w \notin \{u, v\}$, there is some $i \bmod 3$ such that $(u_i, u_{i+1}) \prec (w_i, w_{i+1})$ and $(v_i, v_{i+1}) \prec (w_i, w_{i+1})$, where $\prec$ represents the lexicographic order.

[1]Note that this is called *weak* barycentric representation in [30].

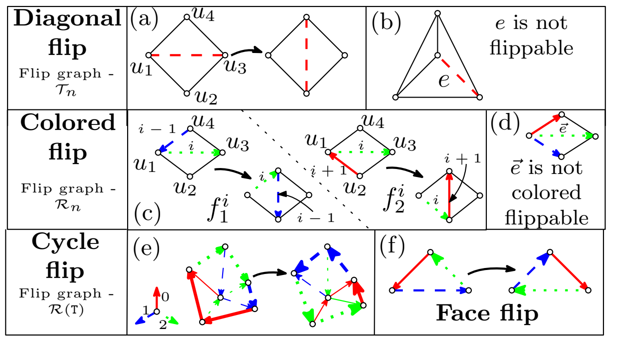


Figure 2: Illustration of different flips.

For each interior vertex u of $\mathtt{T}$ we denote by $P_i(u)$ the path in T_i from u to its root r_i with $i \bmod 3$. For each interior vertex u its paths $P_0(u), P_1(u)$ and $P_2(u)$ divide the triangulation into three disjoint regions $R_0(u), R_1(u)$ and $R_2(u)$ where $R_i(u)$ denotes the region defined by the vertices in path $P_{i+1}(u) \setminus \{u\}$ and the interior vertices enclosed by paths $P_{i-1}(u)$ and $P_{i+1}(u)$. See Fig. 1(c). The following lemma about these regions was shown in [30].

Lemma 1 (Schnyder [30]) *For every different pair of interior vertices u and v of a triangulation it holds that if $v \in R_i(u) \cup P_{i-1}(u)$, then $R_i(v) \subset R_i(u)$.*

Let $|R_i(u)|$ and $|P_i(u)|$ denote the number of vertices in $R_i(u)$ and $P_i(u)$, respectively. Let n be the total number of vertices. Let $f : V(\mathtt{T}) \to \mathbb{R}^3$ be the function defined as follows. For each interior vertex u in $\mathtt{T}$, $f(u) = \frac{1}{n-1}(|R_0(u)|, |R_1(u)|, |R_2(u)|)$, and for each root $r_i \in T_i$, $f(r_i)$ has its ith coordinate equal to $n-1$, its $(i+1)$th coordinate equal to 1 and its $(i+2)$th coordinate equal to 0. Schnyder [30] showed that f defines barycentric coordinates of the vertices of $\mathtt{T}$. Thus, every triangulation admits a barycentric representation that is in correspondence with a Schnyder wood.

We say that a triangulation is a *plane triangulation* if the triangulation is embedded in the plane such that no two edges intersect in their interior. From now onwards, we will refer to a triangulation as a plane triangulation.

3 Flips

In this section, we study the relationship between a *diagonal flip* in triangulations with $n \geq 4$ vertices, a *colored* *flip* in a Schnyder wood, and a *cycle flip* of a Schnyder wood of a triangulation. See Fig. 2.

A *diagonal flip* in a triangulation T is the operation that exchanges the diagonal u_1u_3 of a convex quadrilateral $u_1u_2u_3u_4$ in T by the diagonal u_2u_4. See Fig. 2(a). The *flip graph* $\mathcal{T}_n$ *of triangulations* of n given vertices in the plane is defined as the graph with vertex set defined by all distinct triangulations on n vertices and two vertices of $\mathcal{T}_n$ are adjacent if their corresponding triangulations can be transformed into each other by exactly one diagonal flip. We say that the diagonal u_1u_3 of a quadrilateral $u_1u_2u_3u_4$ in T is *flippable* if the edge u_2u_4 is not in T. See Fig. 2(b).

Wagner [34] showed that the flip graph $\mathcal{T}_n$ is connected: Any triangulation of n vertices can be transformed into another by a finite sequence of diagonal flips. Given that a Schnyder wood is an orientation of the edges of a triangulation, it is natural to ask whether these flips can be extended to Schnyder woods. In other words, whether there exist local functions that transform one Schnyder wood into another. Bonichon et al. [6] defined flips in Schnyder woods that map to diagonal flips in the underlying triangulation.

We refer to edge $\overrightarrow{uv}$ of a Schnyder wood of a triangulation T as the oriented edge uv of T. A *colored flip* in a Schnyder wood of edge $\overrightarrow{uv}$ with respect to $\overrightarrow{wu}$ of the quadrilateral $uwvz$ is the operation that replaces the edges $\overrightarrow{uv}$ and $\overrightarrow{wv}$ in tree T_i and T_j, by the edges $\overrightarrow{uw}$ and $\overrightarrow{wz}$, respectively. There are two types of colored flips denoted f_1^i or f_2^i given in Fig. 2(c).

Notice that a colored flip transforms a Schnyder wood of a T into a Schnyder wood of another triangulation T′. Also, note that there might be edges that are flippable in T that are not colored flippable as shown in Fig. 2(d). However, Bonichon et al. [6] showed that the colored flip graph denoted $\mathcal{R}_n$, is connected. Their proof relies on the fact that the flip graph restricted to colored flips of the type f_1^i defines a bounded poset as does the one restricted to their inverse f_2^i. However, the proof does not imply a bound on the diameter of the graph.

For simplicity, we only refer to a directed cycle as a *cycle* in a directed graph. Brehm [9] defines cycle flips between 3-orientations of a given triangulation, with its corresponding definition for Schnyder wood. A *cycle flip* in a Schnyder wood R is the operation that reverses the orientation of a cycle $\mathcal{C}$ such that if $\mathcal{C}$ is counter-clockwise oriented (resp. clockwise oriented), then:

1. the color of each edge in $\mathcal{C}$ is exchanged by the color succeeding (resp. preceding) its original color,
2. for each edge inside $\mathcal{C}$ the new color is set to be the color preceding (resp. succeeding) its original color, and
3. the color of all other edges are unchanged. See Fig. 2(e).

A *face flip* of a Schnyder wood R is a cycle flip of a cycle of length 3 defined by the edges of a face. See Fig. 2(f).

Note that a cycle flip transforms one Schnyder wood of a triangulation T into another Schnyder wood of T, whereas a colored flip transforms one Schnyder wood of a triangulation T into another Schnyder wood of another triangulation T′. This means that the flip graph of cycle flips has as vertex set the set of all Schnyder woods of a triangulation T. While the colored flip graph corresponds to the vertex set of all possible Schnyder woods of all triangulations of n vertices.

Brehm [9] showed that given a 4-connected triangulation T, the flip graph of face flips $\mathcal{R}(\mathtt{T})$ of the Schnyder woods of T is connected. The proof is obtained by showing that the structure of flipping counter-clockwise faces into clockwise defines a poset, similar to the proof of colored flips.

In this section, we provide a new proof of the connectivity of $\mathcal{R}_n$ using the relation between colored flips and cycle flips. In addition, we prove an upper bound on the diameter of $\mathcal{R}_n$. In order to show that $\mathcal{R}_n$ is connected we divide this section as follows. In Subsection 3.1 we show that for any flippable edge uv in a triangulation T there exists a Schnyder wood R of T with oriented edge uv that admits a colored flip. In Subsection 3.2 we show that any cycle flip in a Schnyder wood R can be obtained by a sequence of a linear number of colored flips. We conclude, using the results from the previous results, that two Schnyder woods R and R' can be transformed into each other by $O(n^2)$ colored flips.

3.1 Diagonal flips and colored flips

In this subsection, we show that for each flippable edge e in a triangulation T, there exists a Schnyder wood of T where the oriented edge e is colored flippable.

Let P be a directed path, we denote by uPv as the subpath in P that goes from u to v.

Lemma 2 *Let* T *be a triangulation and R be a Schnyder wood of* T. *Let uv be a flippable edge in* T *where uv is the diagonal of the quadrilateral $uwvz$. If $\overrightarrow{uv}$ is not colored flippable in R, then there exists a cycle $\mathcal{C}$ in R that passes through either $\overrightarrow{uw}$ or $\overrightarrow{uz}$ in R but avoids $\overrightarrow{uv}$. Moreover, $\mathcal{C}$ can be found in $O(n)$ time.*

Proof. Note that if either the edge $\overrightarrow{wu}$ or $\overrightarrow{zu}$ is in R, then $\overrightarrow{uv}$ is colored flippable in R and the result holds. Thus, let us assume that $\overrightarrow{uw}$ and $\overrightarrow{uz}$ are in R. See Fig. 3(a). Let i mod 3 be the label of $\overrightarrow{uv}$ in R, hence the labels of $\overrightarrow{uz}$ and $\overrightarrow{uw}$ are $i+1$ and $i-1$ module 3, respectively.

Since vertices in the outer face of a Schnyder wood are sinks, u is an interior vertex. Since uv is flippable in T, $uwvz$ is convex. Thus, at least one of w or z is an interior vertex of T. Assume without loss of generality that such vertex is w. Consider the paths $P_{i+1}(w)$ and $P_{i+1}(u)$.

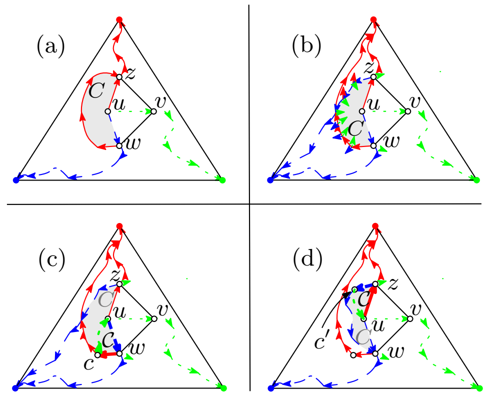


Figure 3: An illustration of Theorem 3: (a) Vertex z is in $P_{i+1}(w)$. (b) Vertex z is not in $P_{i+1}(w)$. (c) The heavier cycle $\mathcal{C}$ when $(P_{i+1}(w) \setminus \{w\}) \cap C \neq \emptyset$. (d) The heavier cycle $\mathcal{C}$ when $(P_{i+1}(w) \setminus \{w\}) \cap C = \emptyset$.

Note that both paths are of length at least 1 since both u and w are interior vertices of T. Since $w \in P_{i-1}(u)$, by Lemma 1 the path $P_{i+1}(w)$ lies in $R_i(u)$. For the same reason, if z is an interior vertex of T, then the path $P_{i-1}(z)$ lies in $R_i(u)$ as well. See Figs. 3(a) and 3(b). Let C be a closed region defined by the following boundary: If $z \in P_{i+1}(w) \cap P_{i+1}(u)$, then $\partial C = wP_{i+1}(w)zuw$. See Fig. 3(a). Otherwise, let $a = P_{i+1}(w) \cap P_{i-1}(z)$. Then, $\partial C = wP_{i+1}(w)aP_{i-1}(z)^{-1}zuw$. See Fig. 3(b).

Note that by definition of Schnyder wood implies the following three facts: **(1)** each vertex in $\partial C \setminus \{u, w, z\}$ has its outgoing edge of T_i inside C. **(2)** each vertex in $(P_{i+1}(w) \cap \partial C) \setminus \{w\}$ and $(P_{i-1}(z) \cap \partial C) \setminus \{z\}$ has its outgoing edge of T_{i-1} and T_{i+1} outside $\mathcal{C}$, respectively. See Fig. 3(b). **(3)** an incoming edge from T_i of an interior vertex x in T lies between its outgoing edges from T_{i+1} and T_{i-1}. Thus, if a vertex in $\partial C \setminus \{u\}$ has an incoming edge from T_i, such edge is not in C. Therefore, for each vertex $x \in \partial C \setminus \{u, z, w\}$ its path $P_i(x)$ passes through u.

Since uv is flippable, wz is not in T. Hence, $\partial C \setminus \{u, z, w\} \neq \emptyset$. Let $\mathcal{C}$ be a cycle in R as follows: If $(P_{i+1}(w) \setminus \{w\}) \cap \partial C \neq \emptyset$, then let $\mathcal{C} = uwcP_i(c)u$, which is a cycle in R with c the first vertex after w in $P_{i+1}(w)$. See Fig. 3(c). Otherwise, let $\mathcal{C} = uzc'P_i(c')u$ be a cycle in R with c' the first vertex after z in $P_{i-1}(z)$. See Fig. 3(d). □

Brehm [9] showed that applying a cycle flip to a Schnyder wood R of a triangulation T transforms R into another Schnyder wood R' of T. Thus, Lemma 2 with Brehm [9] result implies the following.

Theorem 3 *Let e be a flippable edge in a triangulation* T*. Then, there exists a Schnyder wood R of* T *where the oriented edge $\overrightarrow{e}$ in R is colored flippable.*

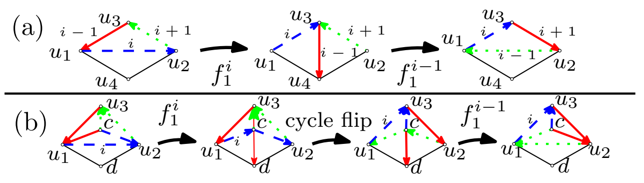


Figure 4: (a) Illustration of Lemma 4. (b) Illustration of Lemma 7.

3.2 Cycle flips and colored flips

In the following, we show that any cycle flip of Schnyder wood results from an $O(n)$ sequence of colored flips, and conclude with an upper bound in the diameter of the colored flip graph.

We say that a triangle $\triangle$ (not necessarily a face) in T is *three-colored* in R if each pair of edges have different colors.

Observation 1 *If a triangle $\triangle$ is a separating cycle in R, then it is a three-colored $\triangle$.*

Fig. 4 illustrates proof of Lemma 4.

Lemma 4 *Let F be an oriented face in a Schnyder wood R. Then, flipping F is equivalent to two colored flips.*

Brehm [9] showed the following lemma for 4-connected triangulations, but the proof shows something stronger.

Lemma 5 *[Brehm [9] Prop. 1.7.3] Let* T *be a triangulation and let $\mathcal{C}$ be a counter-clockwise cycle (resp. clockwise cycle) of length at least 4 in a Schnyder wood R of* T *that contains no separating triangles in $\mathcal{C}$. Then, $\mathcal{C}$ can be cycle flipped by a sequence of face flips of its interior faces where each face is flipped exactly once and it is oriented counter-clockwise (resp. clockwise) before it is flipped.*

Using Lemmas 4 and 5 we obtain the next result.

Corollary 6 *Let* T *be a triangulation and let $\mathcal{C}$ be a cycle of length ≥ 4 in a Schnyder wood R of* T *with m interior faces and it does not contain separating triangles. Then, $\mathcal{C}$ can be cycle flipped by a sequence of $2m$ colored flips.*

Note that from Obs. 1 and the definition of a Schnyder wood, it follows that the interior edges adjacent to the vertices of a cycle that is a separating triangle in R are incoming edges.

Observation 2 *Let $\mathcal{C} = u_1u_2u_3$ be a counter-clockwise (resp. clockwise) cycle that is a separating triangle in a Schnyder wood R. Consider its interior face cu_ju_{j+1} for any j mod 3. Then, R admits an f_1^i (resp. f_2^i) colored flip in the edges cu_j and u_ju_{j+1}. See the first flip in Fig. 4(b).*

Using induction on the number of separating triangles we prove the following lemma.

Lemma 7 *Let $\mathcal{C}$ be a cycle that is a separating triangle in a Schnyder wood R with m interior faces and no interior separating triangles. Then, $\mathcal{C}$ can be cycle flipped by a sequence of $2m$ colored flips.*

Proof. [sketch] First, from Obs. 2 we apply a colored flip to an edge e in $\mathcal{C}$. Then, we observe that there is a cycle $\mathcal{C}'$ oriented as $\mathcal{C}$ and it encloses $m-1$ interior faces of $\mathcal{C}$. Using induction on the number of separating triangles and from Lemmas 4 and 5 we obtain that after $2m-2$ colored flips we obtain a cycle flip of $\mathcal{C}'$. Finally, after applying a colored flip, we obtain a cycle flip of $\mathcal{C}$ which resulted after $2m$ colored flips. Fig. 4(b) illustrates the proof. □

Next, we can obtain the following.

Lemma 8 *Let $\mathcal{C}$ be a cycle with m interior faces in a Schnyder wood R. Then, $\mathcal{C}$ can be cycle flipped by a sequence of $2m$ colored flips.*

Komuro [25] proved that the diameter of $\mathcal{T}_n$ is $O(n)$. [2] Komuro's and previous results imply the desired theorem.

Theorem 9 *The diameter of $\mathcal{R}_n$ is $O(n^2)$.*

Proof. Let R and R' be two different Schnyder woods in $\mathcal{R}_n$ and consider its underlying triangulations $\mathtt{T}$ and $\mathtt{T}'$, respectively. Let $\mathtt{T}''$ be the triangulation with vertices r_0 and r_1 on its outerface adjacent to all the vertices. $\mathtt{T}''$ has a unique Schnyder wood [9]. From Komuro [25], $\mathtt{T}$ and $\mathtt{T}'$ can be transformed into $\mathtt{T}''$ by $O(n)$ diagonal flips. Hence, from Theorem 3 it follows that R and R' can be transformed into each other by a sequence of $O(n)$ colored flips and a cycle flip in between such flips. Since there can be cycles with $O(n)$ interior faces, from Lemma 8 it follows that R and R' can be transformed into each other by $O(n^2)$ colored flips. □

4 Dynamic maintenance

In this section, we study the problem of maintaining a Schnyder wood over a sequence of colored flips.

An *Euler Tour tree*(ETT) is a data structure (similar to the Link/Cut tree [31]) proposed by Henzinger and King [23] that maintains a forest of vertex-disjoint rooted trees with costs in its vertices under two dynamic operations: LINK and CUT (see table below). The ETT supports the operations 1–5 from the table below in worst case $O(\log n)$ time. Consider a data structure of a Schnyder wood R as a set of three ETT trees defined by each tree T_0, T_1, T_2. In each vertex u we store its parent $\text{PARENT}_i(u)$ in T_i for each i mod 3, its initial barycentric coordinates IN-COORDINATES(u) and two costs: $\text{D-COST}_i(u)$ and $\text{R-COST}_i(u)$. Where D-COST_i refers to the distance of u to the root r_i of T_i and the R-COST_i refers to the difference between the initial $(i-1)$th coordinate with the current $(i-1)$th coordinate of u. Precisely, $\text{R-COST}_i(u)$ is the amount that has to be added to the initial $(i-1)$th coordinate and subtracted to the initial $(i+1)$th coordinate. The initial R-COST is 0. We define extra functions for our data structure in lines 7–11 from the table below.

LINK(u, v)	Add edge uv.
PARENT(u)	Return parent of vertex u.
CUT(u)	Delete edge uPARENT(u).
COST(u)	Return current cost in u
T-UPDATECOST(u, x)	Add x to the cost of all vertices in subtree $T(u)$.
LABEL(u, v)	Return label of edge uv.
ORIENTATION(u, v)	Return orientation of edge uv.
IN-COORDINATES(u)	Return initial barycentric coordinates of vertex u.
COORDINATES(u)	Return current barycentric coordinates of u.
FLIP(u, v, w, z)	Apply colored flip to edge $\overrightarrow{uv}$ with respect to $\overrightarrow{wu}$.

The R-COST of each vertex will allow it to maintain its barycentric coordinates. First, consider the labels i and j of $\overrightarrow{uv}$ and $\overrightarrow{wu}$, respectively. We observe that the only change made when applying a colored flip to $\overrightarrow{uv}$ are the paths passing through edges $\overrightarrow{uv}$ and $\overrightarrow{uw}$. These paths are exactly the paths $P_i(x)$ for all $x \in T_i(u)$ and $P_j(y)$ for all $y \in T_j(w)$. Thus, the only vertices changing their regions are the ones in $T_i(u)$ and $T_j(w)$. Moreover, the i-th region of any $x \in T_i(u)$ and the j-th region of any $y \in T_j(w)$ remain unchanged. Thus, the regions $R_{i-1}(x)$ and $R_{i+1}(x)$ exchange elements for all $x \in T_i(u)$. The same applies to elements in $T_j(w)$. We obtain Lemma 10 below. Before that, we need a few definitions.

Let $T_i(u)$ denote the subtree of T_i rooted at an interior vertex u. Let R be a Schnyder wood and let R' be the resulting Schnyder wood when applying an f_1^i (resp. f_2^i) colored flip to $\overrightarrow{uv}$ with respect to edge $\overrightarrow{wu}$ in quadrilateral $uwvz$. Define $c(u) = |R_{i-1}(w)| - |R_{i-1}(u)| + 1$ (resp. $c(u) = |R_{i+1}(u)| - |R_{i+1}(w)| + 1$) and define $c(w)$ as follows: If $\overrightarrow{uz} \in R$, then $c(w) = -1$ (resp. $c(w) = 0$). Otherwise, $c(w) = |R_i(u)| - |R_i(z)|$ (resp. $c(w) = |R_i(z)| - |R_i(u)| - 1$).

Lemma 10 *Let R and R', $\overrightarrow{uv}$ and $\overrightarrow{wu}$, $c(u)$, and $c(w)$ defined as above. Then,*

1. *For all x in $T_i(u)$, $|R'_{i-1}(x)| - |R_{i-1}(x)| = c(u)$ and $|R'_i(x)| = |R_i(x)|$.*
2. *For all y in $T_j(w)$, $|R'_{j-1}(y)| - |R_{j-1}(y)| = c(w)$ and $|R'_j(y)| = |R_j(y)|$.*
3. *The regions of any vertex in $V(R) \setminus (V(T_i(u) \cup T_j(w)))$ remain the same.*

Proof. Note that the only change made when applying a colored flip to $\overrightarrow{uv}$ are the paths passing through edges

[2]The best bound known is in [10] but their procedure might change the outerface.

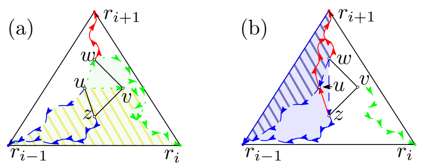


Figure 5: (a) The filled area corresponds to $R'_{i+1}(u)$. (b) The filled area corresponds to $R'_i(w)$.

$\overrightarrow{uv}$ and $\overrightarrow{uw}$. These paths are exactly the paths $P_i(x)$ for all $x \in T_i(u)$ and $P_j(y)$ for all $y \in T_j(w)$. Thus, the regions of any vertex in $V(R) \setminus (V(T_i(u) \cup T_j(w)))$ remain the same.

From Lemma 1 we note that each $x \in T_i(u)$ is in $R_i(u)$. Since the paths in $R_i(u)$ are unchanged in R', it follows that $R'_i(x) = R_i(x)$ for all $x \in T_i(u)$. Similarly, we show that $R'_j(y) = R_j(y)$ for all $y \in T_j(w)$.

Now, we assume $j = i - 1$. Let us show that $|R'_{i-1}(x)| = |R_{i-1}(x)| + c(u)$. Since $u \in P_i(x)$ for each $x \in T_i(u) \setminus \{u\}$, $u \in R_{i-1}(x)$. In addition, since $x \in R_i(u)$, $(R_{i-1}(x) \setminus R_{i-1}(u)) \subset R_i(u)$ which remains unchanged. Hence, $|R'_{i-1}(x)| = |R_{i-1}(x)| - |R_{i-1}(u)| + |R'_{i-1}(u)|$. On the other hand, note that $P_{i-1}(u) \subset P_{i-1}(w)$. Hence, $R_{i+1}(u) \subset R_{i+1}(w)$. Note that $R'_{i+1}(u)$ is given by the region between paths $P_{i-1}(u)$ and $uw \cup P_i(w)$, which is exactly $R_{i+1}(w) \cup \{w\}$. See Fig 5(a). Hence, $|R'_{i+1}(u)| - |R_{i+1}(u)| = |(R_{i+1}(w) \cup \{w\}) \setminus R_{i+1}(u)| = -c(u)$. Therefore, $|R'_{i-1}(x)| = |R_{i-1}(x)| + c(u)$ for all $x \in T_i(u)$.

Finally, let us show that $|R'_{i+1}(y)| = |R_{i+1}(y)| + c(w)$ for all $y \in T_{i+1}(w)$. Since $w \in P_{i-1}(y)$ for each $y \in T_{i-1}(w) \setminus \{w\}$, $R_{i+1}(w) \in R_{i+1}(y)$. In addition, since $y \in R_{i-1}(w)$, $(R_{i+1}(y) \cap R_{i-1}(w)) \subset R_{i+1}(y)$ which remains unchanged. Hence, $|R'_{i+1}(y)| = |R_{i+1}(y)| - |R'_{i+1}(w)| + |R_{i+1}(w)|$.

On the other hand, if $\overrightarrow{uz} \in R$: then $P_{i-1}(w) = (wvz) \cup P_{i-1}(z)$. Since v is the only new vertex in the interior of $R'_{i+1}(w)$ and $|P'_{i+1}(w)| - |P_{i+1}(w)| = -1$, it follows that $|R'_{i+1}(w)| - |R_{i+1}(w)| = -1$. Now, consider the case $\overrightarrow{zu} \in R$: then $P_{i+1}(u) \subset P_{i+1}(z)$. Hence, $R_i(u) \subseteq R_i(z)$. In addition, since $\overrightarrow{wu} \in R$, we have that $P_{i-1}(u) \subset P_{i-1}(w)$ and $R_i(u) \subseteq R_i(w)$. Thus, $R_i(z) \cap R_i(w) = R_i(u)$. Therefore, $|R'_{i+1}(w)| - |R_{i+1}(w)| = -|R_i(z) \setminus R_i(w)| = |R_i(u)| - |R_i(z)| = c(w)$. See Fig 5(b). Therefore, $|R'_{i+1}(y)| = |R_{i+1}(y)| + c(w)$ for every $y \in T_{i-1}(w)$.

The case when $j = i + 1$ is symmetric. □

Note that $|R'_{i+1}(x)|$ and $|R'_{j+1}(y)|$ are given implicitly for each $x \in T_i(u)$ and $y \in T_j(w)$. Now, we obtain the next theorem using the procedures below.

Theorem 11 *A Schnyder wood of a triangulation can be maintained in amortized $O(\log n)$ per* FLIP. *Furthermore, queries* ORIENTATION, LABEL, COORDINATES *and* D-COST$_i$ *can be obtained in $O(\log n)$ amortized time.*

Proof. Consider the procedures defined below. From [23] PARENT and IN-COORDINATES take $O(1)$ time and T-UPDATECOST$_i$ takes $O(\log n)$ amortized time. Since $|R_{i+1}(u)| = n - 1 - |R_i(u)| - |R_{i-1}(u)|$, it follows that COORDINATES(u) is correct. Since IN-COORDINATES takes constant time and R-COST was called exactly three times, it follows that COORDINATES can be obtained in $O(\log n)$ time. Since the functions ORIENTATION and LABEL are calling PARENT at most 6 times, ORIENTATION and LABEL can be obtained in $O(1)$ time. Since D-COST$_i$ and R-COST$_i$ behave as COST from a link/cut tree, then both can be obtained in $O(\log n)$. It remains to analyse FLIP procedure: Note that in line 18 the function removes edges $\overrightarrow{uv}$ and $\overrightarrow{wu}$. In line 19 the new edges $\overrightarrow{uw}$ and $\overrightarrow{wz}$ are added. Thus, FLIP does the desired colored flip. Line 20 changes the R-COST and D-COST for each vertex in the subtree $T_i(u)$ by $c(u)$ and $d(u)$, respectively. Similarly, in the subtree $T_j(w)$ by $c(w)$ and $d(w)$, respectively. From Lemma 10 updated R-COST$_i$ is correct. Therefore, FLIP is correct. Finally, we call exactly 3 times the function COORDINATES, twice each function CUT, LINK and T-UPDATECOST. Each of these functions have amortized cost $O(\log n)$. Hence, FLIP has amortized cost $O(\log n)$. □

```
1: procedure LABEL($u, v$)                        ▷ Returns the label of edge $uv$.
2:     for Each $i$ mod 3 do
3:         if PARENT$_i(u) = v$ then
4:             return $i$
5:         else
6:             if PARENT$_i(v) = u$ then
7:                 return $i$
```

```
1: procedure ORIENTATION($u, v$)                  ▷ Returns orientation of edge $uv$.
2:     Let $b$ = FALSE
3:     for each $i$ mod 3 do
4:         if PARENT$_i(u) = v$ then
5:             let $b$ = TRUE
6:     if $b$ = TRUE then
7:         return $\overrightarrow{uv}$
8:     else
9:         return $\overrightarrow{vu}$
```

```
1: procedure COORDINATES($u$)   ▷ Returns a vector with the barycentric coordinates of $u$
   in current Schnyder wood $R$.
2:     let $(u_0, u_1, u_3)$ = IN-COORDINATES($u$).
3:     for each $i$ mod 3 do
4:         let $c_i$ = R-COST$_i u$.
5:     for each $j$ mod 3 do $u'_j = u_j + \frac{c_{j+1} - c_{j-1}}{n-1}$.
6:     return $(u'_0, u'_1, u'_2)$
```

```
1: procedure FLIP($u, v, w, z$)   ▷ Creates a flip while updates the cost in the subtrees that
   are changed.
2:     Let $i$ = LABEL($u, v$), $j$ = LABEL($u, w$).
3:     let $d(u)$ = D-COST$_i(w)$ − D-COST$_i(u) + 1$ and $d(w)$ = D-COST$_j(z)$ − D-COST$_j(w) + 1$
4:     $(v_0, v_1, v_2)$ = COORDINATES($u$),
5:     $(w_0, w_1, w_2)$ = COORDINATES($w$),
6:     $(z_0, z_1, z_2)$ = COORDINATES($z$).
7:     if $j = i - 1$ mod 3 then
8:         let $c(u) = (n-1)(w_{i-1} - u_{i-1})$.
9:         if ORIENTATION($u, z$) = $\overrightarrow{uz}$ then
10:            let $c(w) = 0$
11:        else
12:            let $c(w) = (n-1)(z_i - u_i)$
13:    else
14:        let $c(u) = (n-1)(u_{i-1} - w_{i-1}) + 1$
15:        if ORIENTATION($u, z$) = $\overrightarrow{uz}$ then
16:            let $c(w) = -1$
17:        else
18:            let $c(w) = (n-1)(u_i - z_i)$
19:    CUT$_i(u)$, CUT$_j(w)$
20:    LINK$_i(u, w)$, LINK$_j(w, z)$
21:    T-UPDATECOST$_i(u, c(u), d(u))$, T-UPDATECOST$_j(w, c(w), d(w))$.
```

References

[1] F. Barrera-Cruz, P. Haxell, and A. Lubiw. Morphing Schnyder drawings of planar triangulations. *Discrete & Computational Geometry*, 61(1):161–184, 2019.

[2] S. Bhore, P. Bose, P. Cano, J. Cardinal, and J. Iacono. Dynamic schnyder woods. *arXiv preprint arXiv:2106.14451*, 2021.

[3] N. Bonichon, S. Felsner, and M. Mosbah. Convex drawings of 3-connected plane graphs. *Algorithmica*, 47(4):399–420, 2007.

[4] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 266–278. Springer, 2010.

[5] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perković. Plane spanners of maximum degree six. In *International Colloquium on Automata, Languages, and Programming*, pages 19–30. Springer, 2010.

[6] N. Bonichon, B. Le Saëc, and M. Mosbah. Wagner's theorem on realizers. In *International Colloquium on Automata, Languages, and Programming*, pages 1043–1053. Springer, 2002.

[7] P. Bose and F. Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009.

[8] P. Bose and S. Verdonschot. A history of flips in combinatorial triangulations. In *Spanish Meeting on Computational Geometry*, pages 29–44. Springer, 2011.

[9] E. Brehm. 3-orientations and Schnyder 3-tree-decompositions. *Master's thesis, FB Mathematik und Informatik, Freie Universität Berlin*, 2000.

[10] J. Cardinal, M. Hoffmann, V. Kusters, C. D. Tóth, and M. Wettstein. Arc diagrams, flip distances, and Hamiltonian triangulations. *Computational Geometry*, 68:206–225, 2018.

[11] L. Castelli Aleardi. Balanced Schnyder woods for planar triangulations: an experimental study with applications to graph drawing and graph separators. In *International Symposium on Graph Drawing and Network Visualization*, pages 114–121. Springer, 2019.

[12] L. Castelli Aleardi and O. Devillers. Array-based compact data structures for triangulations: Practical solutions with theoretical guarantees. *Journal of Computational Geometry*, 9(1):247–289, 2018.

[13] A. Christiansen, J. Holm, E. Rotenberg, and C. Thomassen. Explicit dynamic schnyder woods require linear (amortized) update time. In *European Workshop on Computational Geometry*, 2022.

[14] H. De Fraysseix and P. O. de Mendez. On topological aspects of orientations. *Discrete Mathematics*, 229(1-3):57–72, 2001.

[15] D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer. Separator based sparsification: I. Planarity testing and minimum spanning trees. *Journal of Computer and System Sciences*, 52(1):3–27, 1996.

[16] S. Felsner. Lattice structures from planar graphs. *Electronic Journal of Combinatorics*, pages R15–R15, 2004.

[17] S. Felsner. The order dimension of planar maps revisited. *SIAM Journal on Discrete Mathematics*, 28(3):1093–1101, 2014.

[18] S. Felsner and S. Kappes. Orthogonal surfaces and their CP-orders. *Order*, 25(1):19–47, 2008.

[19] S. Felsner and J. Nilsson. On the order dimension of outerplanar maps. *Order*, 28(3):415–435, 2011.

[20] S. Felsner and W. T. Trotter. Posets and planar graphs. *Journal of Graph Theory*, 49(4):273–284, 2005.

[21] S. Felsner and F. Zickfeld. Schnyder woods and orthogonal surfaces. *Discrete & Computational Geometry*, 40(1):103–126, 2008.

[22] D. Gonçalves and B. Lévêque. Toroidal maps: Schnyder woods, orthogonal surfaces and straight-line representations. *Discrete & Computational Geometry*, 51(1):67–131, 2014.

[23] M. R. Henzinger and V. King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proceedings of the twenty-seventh annual ACM Symposium on Theory of Computing*, pages 519–527, 1995.

[24] J. Holm and E. Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 167–180, 2020.

[25] H. Komuro. The diagonal flips of triangulations on the sphere. *Yokohama mathematical journal*, 44(2):115–122, 1997.

[26] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. *Algorithmica*, 46(3):505–527, 2006.

[27] L. Pournin. The flip-graph of the 4-dimensional cube is connected. *Discrete & Computational Geometry*, 49(3):511–530, 2013.

[28] L. Pournin. The diameter of associahedra. *Advances in Mathematics*, 259:13–42, 2014.

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

[30] W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148, 1990.

[31] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[32] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *Journal of the American Mathematical Society*, 1(3):647–681, 1988.

[33] P. Tetali, A. P. Streib, D. Randall, and S. Miracle. Mixing times of markov chains on 3-orientations of planar triangulations. *Discrete Mathematics & Theoretical Computer Science*, 2012.

[34] K. Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

Appendix

5 Omitted Proofs of Section 3

5.1 Proof of Theorem 3

Proof. Consider an arbitrary Schnyder wood R of $\texttt{T}$ and let $\overrightarrow{e} = \overrightarrow{uv}$ be the orientation of e in R. Let $uwvz$ be the quadrilateral in $\texttt{T}$ with diagonal uv given in counter-clockwise order. If $\overrightarrow{uv}$ admits a colored flip in R, then the statement holds. If $\overrightarrow{uv}$ is not colored flippable, then from Lemma 2 there exists a cycle $\mathcal{C}$ that passes through either $\overrightarrow{uw}$ or $\overrightarrow{uz}$ in R and does not pass through $\overrightarrow{uv}$.

Let R' be the oriented graph when flipping C. Hence, either $\overrightarrow{wu}$ or $\overrightarrow{zu}$ is in R'. In addition, edge $\overrightarrow{uv}$ is in R', since it does not lie in the interior of C in R. By Brehm [9] the directed graph R' is a Schnyder wood of $\texttt{T}$ different from R. Therefore, $\overrightarrow{uv}$ is colored flippable in R'. □

5.2 Proof of Lemma 4

Proof. Consider the oriented face $F = u_1u_2u_3$ in the Schnyder wood R. Let R' be the Schnyder wood obtained when F is face flipped by $F' = u_3u_2u_1$. Without loss of generality assume that F is counter-clockwise oriented and that edge u_1u_2 has label i mod 3 in R. Since F is a three-colored triangle, all u_1, u_2, and u_3 are interior vertices. Hence, u_1u_2 is the diagonal of a quadrilateral $u_1u_4u_2u_3$, see Fig. 4(a). Note that edge u_1u_2 is flippable in $\texttt{T}$: otherwise, either u_2 or u_1 is a vertex of degree 3 in $\texttt{T}$ but with less than three outgoing edges. Which contradicts that R is a Schnyder wood. Since F is counter-clockwise oriented, R admits a f_1^i colored flip in the edge u_1u_2 with respect to u_3u_1. Let R^2 be the resulting Schnyder wood when applying such a flip. Thus, the orientation of $\overrightarrow{u_3u_1}$ changes to $\overrightarrow{u_1u_3}$ and is re-labelled by i. In addition, the edge $\overrightarrow{u_1u_2}$ by $\overrightarrow{v_3v_4}$ and with label $i-1$. Now, note that R^2 admits an f_1^{i-1} colored flip in the edge $\overrightarrow{u_3u_4}$ with respect to $\overrightarrow{v_2v_3}$. Consider the resulting Schnyder wood R^3 when applying such colored flipped in R^2. Then, the edge $\overrightarrow{u_2u_3}$ changes its orientation to $\overrightarrow{u_3u_2}$ and label by $i-1$. In addition, the edge $\overrightarrow{u_3u_4}$ is exchanged by $\overrightarrow{u_2u_1}$ with label $i+1$. Note that the face $u_1u_2u_3$ is now clockwise oriented in R^3 and the label of each edge of F in R is labelled by its successor in R^3. Even more, since no other edge has been changed, it follows that F was face flipped in R^3. Therefore, $R^3 = R'$.

The reverse follows from the fact that f_2 colored flips are the inverse of f_1 colored flips. □

5.3 Proof of Lemma 7

Proof. Let $\mathcal{C} = u_1u_2u_3$ be a counter-clockwise (resp. clockwise) cycle and consider its interior face u_1cu_2 and $\overrightarrow{u_1u_2}$ with label i mod 3. From Obs. 2 we can apply a colored flip to u_1u_2 with respect to $\overrightarrow{cu_1}$ from a quadrilateral u_1cu_2d in $\texttt{T}$. Let R' be the resulting Schnyder wood when applying such colored flip. See Fig. 4(b). Note that the edges $\overrightarrow{u_1c}, \overrightarrow{cu_2}, \overrightarrow{u_2u_3}$ and $\overrightarrow{u_3u_1}$ define a counter-clockwise cycle (resp. clockwise cycle) in R' with $m-1$ interior faces. Let $\mathcal{C}'$ be such cycle. Let R'' be the resulting Schnyder wood when applying a cycle flip to $\mathcal{C}'$. From Corollary 6 it follows that R'' is obtained from R' by a sequence of $2(m-1)$ colored flips.

Note that all interior edges of $\mathcal{C}'$ and edge $\overrightarrow{cu_1}$ are labeled in R'' by its preceding label (resp. succeeding label) in R. Also, edges u_2u_3, u_3u_1 and cu_2 have opposite orientations in R'' and its label in R'' is the succeeding (resp. preceding label) from R. Finally, R'' admits a colored flip in $\overrightarrow{cd}$ with respect to edge $\overrightarrow{u_2c}$. Let R^3 be the resulting Schnyder wood when applying such colored flip. Note that $\overrightarrow{cu_2}$ had label i in R as $\overrightarrow{u_1u_2}$ in R. See Fig. 4(b). Hence, R^3 corresponds to a cycle flip of $\mathcal{C}$ in R. Therefore, $\mathcal{C}$ can be cycle flipped by a sequence of $2m-2+2 = 2m$ colored flips. □

We say that a separating triangle in a cycle $\mathcal{C}$ is *maximal* if it is not contained in another separating triangle contained in $\mathcal{C}$.

5.4 Proof of Lemma 8

Proof. Let us show that $\mathcal{C}$ can be cycle flipped by a sequence of $2m$ colored flips. We proceed by induction on the number of separating cycles t in $\mathcal{C}$.

[Base case] Assume $t = 0$. from Corollary 6 and Lemma 7 the statement holds.

[Inductive Hypothesis] $\mathcal{C}$ can be cycle flipped by a sequence of $2m$ colored flips if it contains $t-1 \geq 0$ separating triangles.

[Inductive step] Assume $\mathcal{C}$ contains t separating triangles. If $\mathcal{C}$ has length at least 4, we denote $R' = R$ and $\mathcal{C}' = \mathcal{C}$ a cycle in R' and $m' = m$. If $\mathcal{C} = u_1u_2u_3$ is of length 3, i.e., is a separating triangle, we define R', $\mathcal{C}'$ and m' as follows. Consider its interior face cu_1u_2 and label i of $\overrightarrow{u_1u_2}$. From Obs. 2 we can apply a colored flip to u_1u_2 from a quadrilateral u_1cu_2d in $\texttt{T}$ such that $\overrightarrow{cu_1}$ is re-oriented to $\overrightarrow{u_1c}$ with label i and $\overrightarrow{u_1u_2}$ is exchanged by $\overrightarrow{cd}$ with same label as $\overrightarrow{cu_1}$ in R. Let R' be the resulting Schnyder wood when applying such colored flip to R. Note that the edges $\overrightarrow{u_1c}, \overrightarrow{cu_2}, \overrightarrow{u_2u_3}$ and $\overrightarrow{u_3u_1}$ define a cycle in R' and is oriented as $\mathcal{C}$. Let $\mathcal{C}'$ be such a cycle. Note that $\mathcal{C}'$ contains $m' = m-1$ interior faces and at most t separating triangles. See Fig. 4(b) for reference.

Let $\triangle_1, \ldots, \triangle_k$ be the maximal separating triangles in $\mathcal{C}'$ with $m_1, \ldots, m_k$ interior faces, respectively. Let $\mathcal{C}''$ be the resulting cycle when removing the interior vertices of each $\triangle_1, \ldots, \triangle_k$. Then, $\mathcal{C}''$ is a cycle of length at least 4 with no separating triangles. By Lemma 5, $\mathcal{C}''$ can be cycle flipped by a sequence of $m' - (\sum_{j=1}^{k} m_j) + k$ face flips. Note that each face flip of a triangle $\triangle_j$ in $\mathcal{C}''$ corresponds to a cycle flip of $\triangle_j$ in $\mathcal{C}'$. Since each $\triangle_j$ contains at most $t-1$ separating triangles, by inductive hypothesis, we obtained that $\mathcal{C}'$ can be cycle flipped by a sequence of $2(m' - \sum_{j=1}^{k} m_j) + 2\sum_{j=1}^{k} m_j = 2m'$ colored flips.

If $\mathcal{C}$ is of length at least 4, the statement holds. Otherwise, since $\mathcal{C}'$ had the same orientation as $\mathcal{C}$, then when applying the cycle flip to $\mathcal{C}'$ we obtained $cu_2u_3u_1$ in the resulting Schnyder wood R'' and R'' admits a colored flip in edges $\overrightarrow{cd}$ and $\overrightarrow{u_2c}$. Let R^3 be the resulting Schnyder wood when applying the colored flip to $\overrightarrow{cd}$. Note that R^3 contains the cycle $u_3u_2u_1$ and edge $\overrightarrow{cu_2}$ with the desired labeling as for a cycle flip. Thus, $\mathcal{C}$ can be cycle flip by a sequence of $2 + 2m' = 2m$ colored flips.

□

Geometric Algorithms for k-NN Poisoning

Diego Ihara Centurion* Karine Chubarian* Bohan Fan* Francesco Sgherzi*
Thiruvenkadam S Radhakrishnan* Anastasios Sidiropoulos* Angelo Straight*

Abstract

We propose a label poisoning attack on geometric data sets against k-nearest neighbor classification. We provide an algorithm that can compute an εn-additive approximation of the optimal poisoning in $n \cdot 2^{2^{O(d+k/\varepsilon)}}$ time for a given data set $X \in \mathbb{R}^d$, where $|X| = n$. Our algorithm achieves its objectives through the application of multi-scale random partitions.

1 Introduction

Recent developments in machine learning have spiked the interest in robustness, leading to several results in *adversarial machine learning* [1, 2, 11]. A central goal in this area is the design of algorithms that are able to impair the performance of traditional learning methods by adversarially perturbing the input [3, 17, 19]. Adversarial attacks can be exploratory, such as evasion attacks, or causative, poisoning the training data to affect the performance of a machine learning algorithm or attack the algorithm itself. Backdoor poisoning is a type of causative adversarial attack, in which the attacker has access to the whole or a portion of the training data that they can perturb. Clean-label poisoning attacks are a type of backdoor poisoning attack that perturb only the features of the training data leaving the labels untouched, so as to make the poison less detectable. In the other end of the spectrum are label poisoning attacks that perturb or flip the training data labels.

Why compute provably nearly-optimal poison attacks? A limitation with current poisoning methods is that it is not possible to adversarially perturb an input so that the performance of *any* algorithm is negatively affected. Moreover, it is generally not clear how to provably compare different poisoning methods. We seek to address these limitations of adversarial machine learning research using tools from computational geometry.

Specifically, we study the following optimization problem: Given some data set, X, compute a small perturbation of X, so that the performance of a specific classifier deteriorates as much as possible. An efficient solution to this optimal poisoning problem can be used to compare the performance of different classification algorithms, as follows. Suppose we want to compare the performance of a collection of classification algorithms, $A_1, ..., A_t$, on some fixed data set X, in the presence of a poisoning attack that produces a bounded perturbation, X', of X. Ideally, we would like to have provable worst-case guarantees on the robustness of $A_1, ..., A_t$. However, such results are often hard to prove rigorously, and thus many existing methods lack such guarantees. Since the poisoned data set X' is unknown, we cannot simply run A_1, ..., A_t on X' and compare the results. Instead, our method allows us to compute from X some poisoned data set, X'', which is provably a nearly-optimal poison against the specific classification task.

1.1 Robustness of k-Nearest Neighbors

We instantiate the above general optimization problem of computing nearly-optimal poison attacks to the specific task of k-nearest neighbor classification. Nearest-neighbor based algorithms are naturally robust due to the presence of an inherent majority voting mechanism. In [12], they are used to provide individual and joint certifications for test predictions in the presence of data poisoning and backdoor attacks. In [16], a defense algorithm is proposed using k-nearest neighbors against label-flipping attacks. However, computing provably nearly-optimal poisoning against such algorithms has not been studied prior to our work. We provide an approximation algorithm that computes an optimal label flipping poisoning attack against k-nearest neighbors that achieve provable guarantees.

1.2 Our Results

We design and analyze poisoning algorithms against k-nearest neighbor classification (k-NN) in the setting of binary label classification. The k-NN classifier is arguably one of the most popular and well-studied methods used in machine learning and geometric data analysis [8]. The classifier works as follows: Given a set of labeled points, $X \subset \mathbb{R}^d$, and some unlabeled $p \in \mathbb{R}^d$, we can compute a label for p by taking the most frequently occurring label in the multiset of labels of the k nearest neighbors of p in X.

We formulate the poisoning problem against k-NN as

*Department of Computer Science, University of Illinois at Chicago, {dihara2, kchuba2, bfan4, fsgher2, tsivap2, sidiropo, astrai3}@uic.edu

follows. Given some set of points, X, with binary labels, and some $m \in \mathbb{N}$, the goal is to flip the labels of at most m points, so that we maximize the number of points in X for which their *predicted* label differs from their true label. We refer to the set of points with flipped labels as an m-poison and define the number of points for which their predicted label differs from the original label as the *corruption*. The following summarizes our results.

Theorem 1 *On input $X \subset \mathbb{R}^d$, with $|X| = n$, and $m \in \mathbb{N}$, Algorithm Poison-k-NN computes a m-poison against k-NN, with expected corruption $\mathsf{OPT}_m(X) - \varepsilon n$, in time $n \cdot 2^{2^{O(d+k/\varepsilon)}}$, where $\mathsf{OPT}_m(X)$ denotes the maximum corruption of any m-poison.*

While the above problem formulation only involves a fully labeled set X, typical tasks involve a training set X_{train} and a test set X_{test}. In order to address this case, we modify the algorithm in Theorem 1 so that it computes a poison of the training set, so that the prediction error on the test set deteriorates as much as possible. This result is summarized in the following.

Theorem 2 *On input $X_{train}, X_{test} \subset \mathbb{R}^d$, with $|X_{train}| = n_{train}$, $|X_{test}| = n_{test}$, and $m \in \mathbb{N}$, Algorithm Poison-k-NN' computes a m-poison against k-NN, with expected corruption $\mathsf{OPT}_m(X_{train}, X_{test}) - \varepsilon n_{test}$, in time $(n_{train} + n_{test}) \cdot 2^{2^{O(d+k/\varepsilon)}}$, where $\mathsf{OPT}_m(X_{train}, X_{test})$ denotes the maximum corruption incurred on X_{test} when all neighbors are chosen from X_{train}, of any m-poison on X_{train}.*

Algorithm Poison-k-NN' is an adaptation of Poison-k-NN, and has a similar analysis. Algorithm Poison-k-NN uses as a subroutine a procedure for computing a random partition of a metric space. The random partition has the property that the diameter of each cluster is upper bounded by some given Lipschitz function, while the probability of two points being separated is upper bounded by a multiple of their distance divided by the cluster diameter (see Section 2 for a formal definition). This is inspired by the multi-scale random partitions invented by Lee and Naor [13] in the context of the Lipschitz extension problem. We believe that our formulation could be of independent interest.

1.3 Related Work

To the best of our knowledge, no prior work tackles the adversarial poisoning of geometric algorithms giving provable bounds. The most traditional poisoning method is the **fsgm** [9], which consists in adding, to each testing sample, noise with the same dimensionality of the input and proportional to the gradient of the cost function in that point. This approach is proven to be the most damaging adversarial example against linear models like logistic regression. However, it is less effective on deep neural networks, as they are able to approximate arbitrary functions [10]. **pgd** [15] improves upon **fsgm** by iteratively looking for better adversarial examples for a given input toward the direction of the increase of the cost function. However, although producing *better* adversarial samples, it still encounters the same drawbacks of **fsgm**.

There are a few existing algorithms that perform label flipping attacks. In [16] they use a greedy algorithm to flip the examples that maximize the error on a validation set, when the classifier is trained on the poisoned dataset, and use k-NN to reassign the label in the training set as the defense against this type of label flipping attacks. [20] model the label attacks as a bilevel optimization problem targeting linear classifiers and also experiment with the transferability of these attacks. Traditional poisoning methods, however, do not offer provable guarantees on the reduction in performance, thus yielding results that are not *problem* dependent but rather *implementation* or *model* dependent [5, 6, 7].

There have also been a few defenses proposed against label flipping attacks. In [18] they propose a pointwise certified defense against adversarially manipulated data up to some "radius of perturbation" through randomized smoothing. Specifically, each prediction is certified robust against a certain number of training label flips.

1.4 Notation

For any $k \in \mathbb{N}$, let $[k] = \{1, \ldots, k\}$. Let $M = (X, \rho)$ be a metric space. For any $X' \subseteq X$, we denote by $\mathsf{diam}_M(X')$ the diameter of X', i.e. $\mathsf{diam}_M(X') = \sup_{x,y \in X'} \rho(x, y)$; we also write $\mathsf{diam}(X')$ when M is clear from the context. For any $x \in X$, $Y \subseteq X$, we write $\delta(x, Y) = \inf_{y \in Y} \rho(x, y)$. For any metric space $M = (X, \rho)$, any finite $Y \subset X$, any $i \in \mathbb{N}$, and any $q \in X$, let $\mathsf{NN}_i(q, Y)$ denote the i-th nearest neighbor of q in Y.

1.5 Organization

The rest of the paper is organized as follows. Section 2 presents our result on random partitions of metric spaces. Section 3 presents our algorithm for poisoning against k-NN classifiers. We conclude and highlight some open problems in Section 4.

2 Random partitions of metric spaces

In this Section, we introduce a random metric space partitioning scheme. The main idea is to partition a given metric space so that the radius of each cluster is bounded by some Lipschitz function, while ensuring that only a small fraction of pairs end up in different clusters, in expectation. This partition is used by our

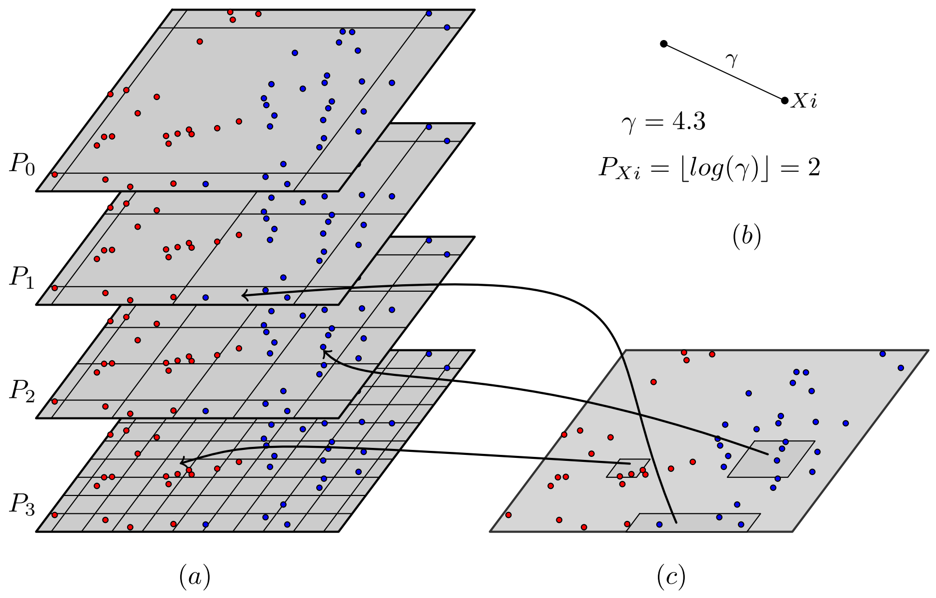


Figure 1: Illustration of the application of the multi-scale random partition approach to a set of points. (a) We begin with a random partition and refine to produce an additional level. (b) The selection of the level depends on the distance to the k-th neighbor. (c) The resulting partition is the union of cells originating at different levels of granularity.

algorithm for partitioning the problem into several subproblems for each cluster.

For any partition P of a ground set Y, and for any $y \in Y$, we denote by $P(y)$ the unique cluster in P that contains y. Let $M = (X, \rho)$ be some metric space. Let P be a random partition of M. For any $\Delta > 0$, we say that P is Δ-bounded if, with probability 1, for all clusters $C \in P$, we have $\mathsf{diam}(C) \leq \Delta$. For any $\beta > 0$, we say that P is β-Lipschitz if for all $x, y \in X$,

$$\Pr[P(x) \neq P(y)] \leq \beta \frac{\rho(x,y)}{\Delta}.$$

The infimum $\beta > 0$ such that for all $\Delta > 0$, M admits a Δ-bounded β-Lipschitz random partition, is referred to as the *modulus of decomposability* of M, and is denoted by β_M.

Lemma 3 ([4]) *For any $d \in \mathbb{N}$, and for subset of d-dimensional Euclidean space, M, we have that $\beta_M = O(\sqrt{d})$.*

The following uses ideas from [13] and [14].

Lemma 4 (Multi-scale random partition) *Let $C > 0$. Let $M = (X, \rho)$ be a metric space, and let $r : X \to \mathbb{R}_{\geq 0}$. Then there exists a random partition P of M, satisfying the following conditions:*

(1) The following statement holds with probability 1: For any $p \in X$,

$$\mathsf{diam}(P(p)) \leq r(p)C^2$$

(2) For any $p, q \in X$,

$$\Pr_P[P(p) \neq P(q)] \leq \left(\frac{2\|r\|_{\mathsf{Lip}}}{\log C} + \beta_M\right) \frac{\rho(p,q)}{r(p)}$$

Moreover, given M as input, the random partition, P can be sampled in time polynomial in $|X|$.

Proof. Let $B = \|r\|_{\mathsf{Lip}}$

For any $i \in \mathbb{Z}$, let P_i be a C^i-bounded β_M-Lipschitz random partition of M. Thus, each point $x \in X$ is mapped to some cluster in each P_i. We construct P by assigning each $x \in X$ to a single one of these clusters. We first sample $\alpha \in [0,1]$, uniformly at random. Then, for each $x \in X$, we assign x to the cluster $P_{\lceil \alpha + \log_C(r(x)) \rceil}(x)$. This concludes the construction of P. It remains to show that the assertion is satisfied.

For any $p \in X$, we have that $P(p) \subseteq P_i(p)$, where $i = \lceil \alpha + \log_C(r(x)) \rceil \leq 2 + \log_C(r(x))$. Since P_i is C^i-bounded, it follows that

$$\begin{aligned} \mathsf{diam}(P(p)) &\leq \mathsf{diam}(P_i(p)) && (P(p) \subseteq P_i(p)) \\ &\leq C^i && (P_i \text{ is } C^i\text{-bounded}) \\ &\leq C^{2+\log_C(r(p))} \\ &= r(p)C^2, \end{aligned}$$

with probability 1, which establishes part (1) of the assertion.

It remains to establish part (2). Let $p, q \in X$. We may assume, without loss of generality, that $0 < r(p) \leq$

$r(q)$. Let $\mathcal{E}_1$ be the event that $\lceil \alpha + \log_C(r(p)) \rceil \neq \lceil \alpha + \log_C(r(q)) \rceil$. We have

$$\begin{aligned}
\Pr[\mathcal{E}_1] &= \Pr[\lceil \alpha + \log_C(r(p)) \rceil \neq \lceil \alpha + \log_C(r(q)) \rceil] \\
&\leq |\log_C(r(p)) - \log_C(r(q))| \\
&= \log_C \frac{r(q)}{r(p)} \\
&= \left(\log \frac{r(q)}{r(p)}\right) / (\log C) \\
&\leq (1/\log C) \log \frac{r(p) + B \cdot \rho(p,q)}{r(p)} \qquad (\|r\|_{\mathsf{Lip}} = B) \\
&= (1/\log C) \log \left(1 + \frac{B \cdot \rho(p,q)}{r(p)}\right) \\
&\leq (1/\log C) 2B \frac{\rho(p,q)}{r(p)}. \qquad (1)
\end{aligned}$$

Conditioned on $\neg\mathcal{E}_1$, there exists $t \in \mathbb{Z}$, such that $t = \lceil \alpha + \log_C(r(p)) \rceil = \lceil \alpha + \log_C(r(q)) \rceil$; let $\mathcal{E}_2$ be the event that $P_t(p) \neq P_t(q)$. Since P_t is C^t-bounded β_M-Lipschitz, it follows that

$$\Pr[\mathcal{E}_2] \leq \beta_M \frac{\rho(p,q)}{C^t} \leq \beta_M \frac{\rho(p,q)}{C^{\log_C(r(p))}} = \beta_M \frac{\rho(p,q)}{r(p)}. \qquad (2)$$

By (1) and (2) we obtain that

$$\Pr[P(p) \neq P(q)] \leq ((1/\log C) 2B + \beta_M) \frac{\rho(p,q)}{r(p)},$$

which establishes part (2) of the assertion, and concludes the proof. □

Figure 1 illustrates the partitioning process.

3 Poisoning k-NN

In this Section, we describe our poisoning algorithm, which is our main result.

Let $d \in \mathbb{N}$, and let $X \subset \mathbb{R}^d$. Let $\mathsf{label} : X \to \{1, 2\}$, and let $k \in \mathbb{N}$ be odd, with $k \leq n$. For any $p \in \mathbb{R}^d$, for any $i \in [k]$, let $\Gamma_i(p)$ be an i-th nearest neighbor of p, breaking ties arbitrarily, and let

$$\gamma_i(p) = \|p - \Gamma_i(p)\|_2.$$

We write $\gamma(p) = \gamma_k(p)$.

Lemma 5 *The function $\gamma : \mathbb{R}^d \to \mathbb{R}$ is 1-Lipschitz.*

Proof. WLOG, let $\gamma(p) \geq \gamma(q)$. It is sufficient to prove that $\gamma(p) \leq \|p - q\| + \gamma(q)$, which means $\Gamma_k(p)$ is within distance $\|p - q\| + \gamma(q)$ from p. Now consider two cases:

Case 1: $\Gamma_k(p)$ falls in $\mathsf{ball}(q, \gamma(q))$. By triangle inequality, $\gamma(p) \leq \|p - q\| + \|\Gamma_k(p) - q\| \leq \|p - q\| + \gamma(q)$.

Case 2: $\Gamma_k(p)$ falls outside of $\mathsf{ball}(q, \gamma(q))$. If $\gamma(p) > \|p - q\| + \gamma(q)$, then all the k-nearest neighbour of q are closer to p than $\Gamma_k(p)$, which is a contradiction.

□

Lemma 6 (Euclidean multi-scale random partition) *Let $\varepsilon > 0$, there exists a random partition P of X, satisfying the following conditions:*

(1) The following statement holds with probability 1: For any $p \in X$,

$$\mathsf{diam}(P(p)) \leq \gamma(p) 2^{8k/\varepsilon} O(\sqrt{d})$$

(2) For any $p, q \in X$,

$$\Pr[P(p) \neq P(q)] \leq \frac{\varepsilon \|p - q\|_2}{k\gamma(p)}.$$

Moreover, P can be sampled in time polynomial in $|X|$.

Proof. Let $M = (X, \rho)$ be the metric space obtained by setting ρ to be the Euclidean metric. By Lemma 3 we have $\beta_M = O(\sqrt{d})$. Let P be the random partition of X obtained by applying Lemma 4, setting $\gamma : X \to \mathbb{R}_{\geq 0}$ where $r = B\gamma$, with $B = 2k\beta_M/\varepsilon$, and $C = 2^{4k/\varepsilon}$. By Lemma 5 we have that $\|\gamma\|_{\mathsf{Lip}} = 1$, and thus $\|r\|_{\mathsf{Lip}} = \|B\gamma\|_{\mathsf{Lip}} = B\|\gamma\|_{\mathsf{Lip}} = B$. The assertion now follows by straightforward substitution on Lemma 4. □

Lemma 7 *Let $h > 0$, and let $A \subset \mathbb{R}^d$, such that for all $p \in A$, we have $\mathsf{diam}(A) \leq h \cdot \gamma(p)$. Then, $|X \cap A| = k \cdot h^{d+O(1)}$.*

Proof. For any $p \in \mathbb{R}^d$, we have that $\gamma(p)$ is the distance between p and k-th nearest neighbor of p in X. It follows that the interior of $\mathsf{ball}(p, \gamma(p))$ contains at most k points in X (it contains at most $k - 1$ points in X if $p \notin X$). In particular, the (closed) ball $\mathsf{ball}(p, \gamma(p)/2)$ contains at most k points in X. Let

$$r^* = \inf_{p \in A} \gamma(p).$$

It follows that for all $p \in A$,

$$|X \cap \mathsf{ball}(p, r^*/2)| \leq k. \qquad (3)$$

We have by the assumption that $\mathsf{diam}(A) \leq h \cdot r^*$, and thus $A \subseteq \mathsf{ball}(p^*, R^*)$, for some $p^* \in A$, and some $R^* = 2h \cdot r^*$. For any $0 < \alpha < \beta$, we have that any ball of radius β in $\mathbb{R}^d$ can be covered by at most $O(\beta/\alpha)^d = (\beta/\alpha)^{d+O(1)}$ balls of radius α. Therefore, A can be covered by a set of at most $(R^*/r^*)^{d+O(1)} = h^{d+O(1)}$ balls of radius $r^*/2$. Combining with (3) it follows that

$$|X \cap A| = k \cdot h^{d+O(1)},$$

which concludes the proof. □

3.1 The main poisoning algorithm

We are now ready to describe the main poisoning algorithm against k-NN. For any finite $Y \subset \mathbb{R}^d$, and for any integer $i \geq 0$, let $\texttt{OPT}_i(X)$ be the maximum corruption that can be achieved for X with a poison set of size at most i. Let *corruption*(X, Y) be the corruption of poisoning X by flipping labels of point set $Y \subset X$. Now we describe our poisoning algorithm.

Algorithm Poison-k-NN for k-NN Poisoning: The input consists of $X \subset \mathbb{R}^d$, with $|X| = n$, and $\mathsf{label} : X \to \{1, 2\}$.

Step 1. Sample the random partition P according to the algorithm in Lemma 6.

Step 2. For any cluster $C \subset X$ in P, by Lemma 7 we have that $|C| = k \cdot (\sqrt{d}2^{8k/\varepsilon})^{d+O(1)} = k \cdot 2^{(d+O(1))8k/\varepsilon}$. For any $i \in \{1, \ldots, m\}$, we compute an optimal poisoning, $S_{C,i} \subseteq C$, for C with i poison points via brute-force enumeration. Each solution can be uniquely determined by selecting the i points for which we flip their label. Thus, the number of possible solutions is at most $2^{|C|} = 2^{k \cdot 2^{(d+O(1))8k/\varepsilon}}$. The enumeration can thus be done in time $2^{k \cdot 2^{(d+O(1))8k/\varepsilon}}$, for each cluster in P. Since there are at most n clusters, the total time is $n \cdot 2^{k \cdot 2^{(d+O(1))8k/\varepsilon}} = n \cdot 2^{2^{O(d+k/\varepsilon)}}$.

Step 3. We next combine the partial solutions computed in the previous step to obtain a solution for the whole pointset. This is done via dynamic programming, as follows. We order the clusters in P arbitrarily, as $P = \{C_1, \ldots, C_{|P|}\}$. For any $i \in \{0, \ldots, |P|\}$, $j \in \{1, \ldots, m\}$, let

$$A_{i,j} = \texttt{OPT}_j(C_1 \cup \ldots \cup C_i).$$

We can compute $A_{i,j}$ via dynamic programming using the formula

$$A_{i,j} = \begin{cases} \max\limits_{t \in [j]} \left(A_{i-1,t} + \mathsf{corruption}(C_i, S_{C_i, j-t})\right) & \text{if } i > 0 \\ 0 & \text{otherwise} \end{cases}$$

The size of the dynamic programming table is $O(|P| \cdot m) = O(nm)$. The same recursion can also be used to compute an optimal k-poison, Y, for $C_1 \cup \ldots \cup C_{|P|}$. The algorithm terminates and outputs Y as the final poison for X.

Proof. [Correctness of Dynamic Programming] By definition, $A_{i,j}$ is the optimal(maximum) corruption with j poison points on the first i clusters $(C_1 \cup \ldots \cup C_i)$. $A_{i,0} = 0$ for all i. Suppose the solution $A_{i-1,j}$ is correct, then $A_{i,j}$ is the maximum of optimal corruption for poisoning the first $i-1$ clusters with t points, plus the corruption using the remaining $j - t$ points on i-th cluster. ☐

Lemma 8 $E[\mathsf{corruption}(X, Y)] \geq \mathit{OPT}_m(X) - \varepsilon n.$

Proof. Let $Z \subseteq X$ be an optimal k-poison for X. Recall that P is the random partition sampled in Step 1.

For any $x \in X$, $i \in \mathbb{N}$, let $\mathcal{E}_{x,i}$ be the event that the cluster of P containing x, does not contain the i-nearest neighbors of x in X; i.e. $\mathsf{NN}_i(x, X) \notin P(x)$. Let also $\mathcal{E}_x$ be the event that the cluster of P containing x, does not contain all of the k-nearest neighbors of x in X; i.e.

$$\mathcal{E}_x = \mathcal{E}_{x,1} \vee \ldots \vee \mathcal{E}_{x,k}.$$

Thus

$$\begin{aligned} \Pr[\mathcal{E}_x] &= \Pr[\mathcal{E}_{x,1} \vee \ldots \vee \mathcal{E}_{x,k}] \\ &\leq \sum_{i=1}^{k} \Pr[\mathcal{E}_{x,i}] && \text{(union bound)} \\ &= \sum_{i=1}^{k} \Pr[P(x) \neq P(\mathsf{NN}_i(x))] \\ &\leq \sum_{i=1}^{k} \frac{\varepsilon \|x - \mathsf{NN}_i(x)\|_2}{k\gamma(p)} && \text{(Lemma 6)} \\ &\leq k \frac{\varepsilon \|x - \mathsf{NN}_k(x)\|_2}{k\gamma(p)} \\ &= \varepsilon && (4) \end{aligned}$$

Let

$$X' = \{x \in X : \{\mathsf{NN}_1(x), \ldots, \mathsf{NN}_k(x)\} \not\subseteq P(x)\}.$$

By (4) and the linearity of expectation, it follows that

$$E[|X'|] = \sum_{x \in |X|} Pr[\mathcal{E}_x] \leq \varepsilon |X| = \varepsilon n. \qquad (5)$$

Let Y be the poison that the algorithm returns. Note that $X \setminus X' = C_1 \cup \ldots \cup C_{|P|}$. For any $x \in X \setminus X'$, if Y corrupts x in $X \setminus X'$, then it must also corrupt x in X (since, by the definition of X, all k-nearest neighbors of x are in $X \setminus X'$). Thus, $\texttt{OPT}_m(X) \geq \texttt{OPT}_m(X \setminus X') \geq \texttt{OPT}_m(X) - |X'|$, and moreover,

$$\begin{aligned} \mathsf{corruption}(X, Y) &\geq \mathsf{corruption}(X \setminus X', Y) \\ &= \texttt{OPT}_m(X \setminus X') \\ &\quad \text{(by the dynamic program)} \\ &\geq \texttt{OPT}_m(X) - |X'|. \end{aligned}$$

Combining with (5) and the linearity of expectation we get $E[\mathsf{corruption}(X, Y)] \geq \texttt{OPT}_m(X) - E[|X'|] \geq \texttt{OPT}_m(X) - \varepsilon n$, which concludes the proof. ☐

Proof. [Proof of Theorem 1] The bound on the corruption follows by Lemma 8. The running time is dominated by Step 2, which takes time $n \cdot 2^{2^{O(d+k/\varepsilon)}}$. ☐

The proof of Theorem 2 follows a similar argument as the proof of Theorem 1, and will be provided in the appendix.

4 Conclusion

We have introduced an approximation algorithm along with provable guarantees for a label flipping poisoning attack against the geometric classification task of k-nearest neighbors. Our poisoning framework, specifically the application of approximation algorithms using random metric partitions could also be extended to propose similar defense algorithms.

References

[1] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang. Recent advances in adversarial training for adversarial robustness. In Z. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4312–4321. ijcai.org, 2021.

[2] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognit.*, 84:317–331, 2018.

[3] N. Carlini. Poisoning the unlabeled dataset of semi-supervised learning. In M. Bailey and R. Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1577–1592. USENIX Association, 2021.

[4] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 379–388. IEEE, 1998.

[5] A. Chhabra, A. Roy, and P. Mohapatra. Suspicion-free adversarial attacks on clustering algorithms. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, February 7-12, 2020*, pages 3625–3632. AAAI Press, 2020.

[6] A. Chhabra, A. Singla, and P. Mohapatra. Fairness degrading adversarial attacks against clustering algorithms. *CoRR*, abs/2110.12020, 2021.

[7] A. E. Cinà, A. Torcinovich, and M. Pelillo. A black-box adversarial attack for poisoning clustering. *Pattern Recognition*, 122:108306, 2022.

[8] E. Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[9] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[10] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[11] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

[12] J. Jia, X. Cao, and N. Z. Gong. Certified robustness of nearest neighbors against data poisoning attacks. *CoRR*, abs/2012.03765, 2020.

[13] J. R. Lee and A. Naor. Extending lipschitz functions via random metric partitions. *Inventiones mathematicae*, 160(1):59–95, 2005.

[14] J. R. Lee and A. Sidiropoulos. On the geometry of graphs with a forbidden minor. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 245–254, 2009.

[15] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[16] A. Paudice, L. Muñoz-González, and E. C. Lupu. Label sanitization against label flipping poisoning attacks. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 5–15. Springer, 2018.

[17] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas. A taxonomy and survey of attacks against machine learning. *Comput. Sci. Rev.*, 34, 2019.

[18] E. Rosenfeld, E. Winston, P. Ravikumar, and Z. Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, pages 8230–8241. PMLR, 2020.

[19] K. Sadeghi, A. Banerjee, and S. K. S. Gupta. A system-driven taxonomy of attacks and defenses in adversarial machine learning. *IEEE Trans. Emerg. Top. Comput. Intell.*, 4(4):450–467, 2020.

[20] M. Zhao, B. An, W. Gao, and T. Zhang. Efficient label contamination attacks against black-box learning models. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3945–3951, 2017.

Appendix

4.1 Poisoning k-NN: with train and test datasets

In this section, we extend our poisoning algorithm to the case with train and test datasets. We provide proof of Theorem 2 by extending the lemmas from the paper.

Theorem 2 *On input $X_{train}, X_{test} \subset \mathbb{R}^d$, with $|X_{train}| = n_{train}$, $|X_{test}| = n_{test}$, and $m \in \mathbb{N}$, Algorithm Poison-k-NN' computes a m-poison against k-NN, with expected corruption $\texttt{OPT}_m(X_{train}, X_{test}) - \varepsilon n$, in time $(n_{train} + n_{test}) \cdot 2^{2^{O(d+k/\varepsilon)}}$, where $\texttt{OPT}_m(X_{train}, X_{test})$ denotes the maximum corruption incurred on X_{test} when all neighbors are chosen from X_{train}, of any m-poison on X_{train}.*

Lemmas 3 and 4 hold without any modifications.

We now define $\gamma_i(p)$ as follows:

For any $p \in \mathbb{R}^d$, for any $i \in [k]$, let $\Gamma_i(p)$ be the i-th nearest neighbor of p in X_{train}, breaking ties arbitrarily, and let

$$\gamma_i(p) = \|p - \Gamma_i(p)\|_2.$$

We denote $\gamma_k(p)$ as $\gamma(p)$.

Lemma 9 (Euclidean multi-scale random partition) *Let $\varepsilon > 0$, there exists a random partition P of X_{train}, satisfying the following conditions:*

(1) The following statement holds with probability 1: For any $p \in X_{train}$,

$$\mathsf{diam}(P(p)) \leq \gamma(p) 2^{8k/\varepsilon} O(\sqrt{d})$$

(2) For any $p, q \in \mathbb{R}^d$,

$$\Pr[P(p) \neq P(q)] \leq \frac{\varepsilon \|p-q\|_2}{k\gamma(p)}.$$

Lemma 10 *Let $h > 0$, and let $A \subset \mathbb{R}^d$, such that for all $p \in A$, we have $\mathsf{diam}(A) \leq h \cdot \gamma(p)$. Then, $|X_{train} \cap A| = k \cdot h^{d+O(1)}$.*

Proof. For any $p \in \mathbb{R}^d$, we have that $\gamma(p)$ is the distance between p and k-th nearest neighbor of p in X_{train}. It follows that the interior of $\mathsf{ball}(p, \gamma(p))$ contains at most k points in X_{train} (it contains at most $k-1$ points in X_{train} if $p \notin X_{train}$). In particular, the (closed) ball $\mathsf{ball}(p, \gamma(p)/2)$ contains at most k points in X_{train}. Let

$$r^* = \inf_{p \in A} \gamma(p).$$

It follows that for all $p \in A$,

$$|X_{train} \cap \mathsf{ball}(p, r^*/2)| \leq k. \tag{6}$$

We have by the assumption that $\mathsf{diam}(A) \leq h \cdot r^*$, and thus $A \subseteq \mathsf{ball}(p^*, R^*)$, for some $p^* \in A$, and some $R^* = 2h \cdot r^*$. For any $0 < \alpha < \beta$, we have that any ball of radius β in $\mathbb{R}^d$ can be covered by at most $O(\beta/\alpha)^d = (\beta/\alpha)^{d+O(1)}$ balls of radius α. Therefore, A can be covered by a set of at most $(R^*/r^*)^{d+O(1)} = h^{d+O(1)}$ balls of radius $r^*/2$. Combining with (6) it follows that

$$|X_{train} \cap A| = k \cdot h^{d+O(1)},$$

which concludes the proof. □

Proof. Let $M = (X, \rho)$ be the metric space obtained by setting ρ to be the Euclidean metric. By Lemma 3 we have $\beta_M = O(\sqrt{d})$. Let P be the random partition of X_{train} obtained by applying Lemma 4, setting $\gamma : X \to \mathbb{R}_{\geq 0}$ where $r = B\gamma$, with $B = 2k\beta_M/\varepsilon$, and $C = 2^{4k/\varepsilon}$. By Lemma 5 we have that $\|\gamma\|_{\mathsf{Lip}} = 1$, and thus $\|r\|_{\mathsf{Lip}} = \|B\gamma\|_{\mathsf{Lip}} = B\|\gamma\|_{\mathsf{Lip}} = B$. The assertion now follows by straightforward substitution on Lemma 4. □

4.2 The poisoning algorithm with train and test datasets

In this section, we describe the poisoning algorithm that will be used to obtain an m-poison with the guarantees of Theorem 2. This follows the algorithm 3.1 with three major differences:

- The $\gamma_i(p)$ function is only defined with respect to the points within X_{train}
- The random partition in Step 1 is only on X_{train}
- The corruption in Step 2 is measured only on X_{test} for the test points that fall within the same cluster

For any finite $Y \subset \mathbb{R}^d$, and any integer $i \geq 0$, let $\texttt{OPT}_i(X_{train}, X_{test})$ be the maximum corruption that can be achieved for X_{train}, X_{test} with a poison set size of at most i. Let *corruption*(X_{train}, X_{test}, Y) be the corruption of X_{test} by flipping the labels of $Y \subseteq X_{train}$.

Algorithm Poison-k-NN for k-NN Poisoning with Train-Test: The input consists of X_{train} and $X_{test} \subset \mathbb{R}^d$, with $|X_{train}| = n_{train}$, $|X_{test}| = n_{test}$ and a map $\mathsf{label} : X \to \{1, 2\}$ that maps X_{train} and X_{test} to their corresponding labels.

Step 1. Sample the random partition of X_{train} - P according to the algorithm in Lemma 9.

Step 2. For any cluster $C \subset X_{train}$ in P, by Lemma 10 we have that $|C| = k \cdot (\sqrt{d}2^{8k/\varepsilon})^{d+O(1)} = k \cdot 2^{(d+O(1))8k/\varepsilon}$. For any $i \in \{1, \ldots, m\}$, we compute an optimal poisoning, $S_{C,i} \subseteq C$, for C with i poison points via brute-force enumeration. Each solution can be uniquely determined by selecting the i points for which we flip their label. Thus, the number of possible solutions is at most $2^{|C|} = 2^{k \cdot 2^{(d+O(1))8k/\varepsilon}}$. The enumeration can thus be done in time $2^{k \cdot 2^{(d+O(1))8k/\varepsilon}}$, for each cluster in P_X. For each possible solution, we also measure the corruption of the poisoning on the points in test set that fall within the same cluster, which takes $O(n_{test})$ time. Since there are at most n_{train} clusters, the total time is $(n_{train} + n_{tst}) \cdot 2^{k \cdot 2^{(d+O(1))8k/\varepsilon}} = (n_{train} + n_{test}) \cdot 2^{2^{O(d+k/\varepsilon)}}$.

Step 3. We next combine the partial solutions computed in the previous step to obtain a solution for the whole pointset. This is done via dynamic programming, as follows. We order the clusters in P arbitrarily, as $P = \{C_1, \ldots, C_{|P|}\}$. For any $i \in \{0, \ldots, |P|\}$, $j \in \{1, \ldots, m\}$, let

$$A_{i,j} = \texttt{OPT}_j(C_1 \cup \ldots \cup C_i).$$

We can compute $A_{i,j}$ via dynamic programming using the formula

$$A_{i,j} = \begin{cases} \max_{t\in[j]} \left(A_{i-1,t} + \mathsf{corruption}(C_i, S_{C_i,j-t})\right) & \text{if } i > 0 \\ 0 & \text{otherwise} \end{cases}$$

The size of the dynamic programming table is $O(|P| \cdot m) = O(nm)$. The same recursion can also be used to compute an optimal k-poison, Y, for $C_1 \cup \ldots \cup C_{|P|}$. The algorithm terminates and outputs Y as the final poison for X.

Lemma 11 $E[\mathsf{corruption}(X_{train}, X_{test}, Y)] \geq \mathtt{OPT}_m(X_{train}, X_{test}) - \varepsilon n_{test}$.

Proof. Let $Z \subseteq X_{train}$ be an optimal k-poison for X_{test}. Recall that P is the random partition sampled at Step 1.

For any $x \in X_{test}$, $i \in \mathbb{N}$, let $\mathcal{E}_{x,i}$ be the event that the cluster of P containing x, does not contain the i_{th}-nearest neighbor of x in X_{train}; i.e. $\mathsf{NN}_i(x) \notin P(x)$. Let also $\mathcal{E}_x$ be the event that the cluster of P containing x, does not contain all of the k-nearest neighbors of x in X_{train}; i.e.

$$\mathcal{E}_x = \mathcal{E}_{x,1} \vee \ldots \vee \mathcal{E}_{x,k}.$$

Thus

$$\begin{aligned} \Pr[\mathcal{E}_x] &= \Pr[\mathcal{E}_{x,1} \vee \ldots \vee \mathcal{E}_{x,k}] \\ &\leq \sum_{i=1}^{k} \Pr[\mathcal{E}_{x,i}] && \text{(union bound)} \\ &= \sum_{i=1}^{k} \Pr[P(x) \neq P(\mathsf{NN}_i(x))] \\ &\leq \sum_{i=1}^{k} \frac{\varepsilon\|x - \mathsf{NN}_i(x)\|_2}{k\gamma(p)} && \text{(Lemma 9)} \\ &\leq k\frac{\varepsilon\|x - \mathsf{NN}_k(x)\|_2}{k\gamma(p)} \\ &= \varepsilon && (7) \end{aligned}$$

Let

$$X' = \{x \in X_{test} : \{\mathsf{NN}_1(x), \ldots, \mathsf{NN}_k(x)\} \not\subseteq P(x)\}.$$

By (7) and the linearity of expectation, it follows that

$$E[|X'|] = \sum_{x\in|X_{test}|} Pr[\mathcal{E}_x] \leq \varepsilon|X_{test}| = \varepsilon n_{test}. \qquad (8)$$

From (8), it follows that,

$$\begin{aligned} \mathsf{corruption}(X_{train}, X_{test}, Y) &\geq \mathsf{corruption}(X_{train}, X_{test} \setminus X', Y) \\ &= \mathtt{OPT}_m(X_{train}, X_{test} \setminus X') \\ &\quad \text{(by the dynamic program)} \\ &\geq \mathtt{OPT}_m(X_{train}, X_{test}) - |X'|. \end{aligned}$$

Combining with (8) and by linearity of expectation we get $E[\mathsf{corruption}(X, Y)] \geq \mathtt{OPT}_m(X) - E[|X'|] \geq \mathtt{OPT}_m(X) - \varepsilon n_{test}$, which concludes the proof. □

Proof. [Proof of Theorem 2] The bound on the corruption on the test set follows Lemma 11. The running time is dominated by Step 2 of 4.2 which takes time $(n_{train} + n_{test}) \cdot 2^{2^{O(d+k/\varepsilon)}}$.

□

Reducing Nearest Neighbor Training Sets Optimally and Exactly*

Josiah Rohrer† Simon Weber‡

Abstract

In nearest-neighbor classification, a *training set* P of points in $\mathbb{R}^d$ with given classification is used to classify every point in $\mathbb{R}^d$: Every point gets the same classification as its nearest neighbor in P. Recently, Eppstein [SOSA'22] developed an algorithm to detect the *relevant* training points, those points $p \in P$ such that P and $P \setminus \{p\}$ induce different classifications. We investigate the problem of finding the *minimum cardinality reduced training set* $P' \subseteq P$ such that P and P' induce the same classification. We show that if P is in general position, the set of relevant points is such a minimum cardinality reduced training set. Furthermore, we show that finding a minimum cardinality reduced training set for possibly degenerate P is in P for $d = 1$, and NP-complete for $d \geq 2$.

1 Introduction

While it is one of the oldest and simplest to describe classification techniques, *nearest-neighbor classification* [12] is still a widely-used method in supervised learning. A *training set* P consisting of data points in $\mathbb{R}^d$ labelled with their known classifications is used to classify new points in $\mathbb{R}^d \setminus P$. A point q gets the same classification as its nearest neighbor in P (ties are either broken by some fixed rule or the point gets multiple classifications).

There are many variations of nearest-neighbor classification, such as k-nearest neighbor [12], where a point gets the majority classification among its k nearest neighbors, and approximate versions of nearest neighbor [27]. In this paper we only consider the basic version described above.

Nearest neighbor classification and the need to implement it efficiently has motivated many concepts in computational geometry. Voronoi diagrams describe the decomposition of $\mathbb{R}^d$ into cells with the same nearest neighbor, and thus the same nearest neighbor classification [4]. They have been extended to higher-order Voronoi diagrams [4], which analogously describe the cells with the same k nearest neighbors. Much research has gone into efficiently computing Voronoi diagrams [14, 20, 32] as well as point-location techniques to locate the cell of a Voronoi diagram containing a given query point [20, 28]. Any technique based on explicitly storing or computing the Voronoi diagram of the training set is infeasible for higher-dimensional data, since the complexity of the Voronoi diagram of n points in dimension d can reach $\Theta(n^{\lceil d/2 \rceil})$ [30]. For moderate and high dimensions, various methods for approximate nearest neighbor searching have been developed, such as quadtree-based data structures [3, 2, 9, 16] and locality-sensitive hashing [1, 13, 19, 24, 27]. These methods avoid the exponential dependency on d, but are still only marginally better than naively computing the nearest neighbor of a query point by searching through the complete training set.

Instead of improving nearest neighbor algorithms and data structures, significant time and storage can be saved by reducing the size of the training set. A common approach to reducing the training set in a *lossless* manner (without changing the classification of any query point) is to remove all non-relevant points. A *relevant point* (sometimes also called *border point*) is a point whose individual omission changes the classification [11]. One can show that removing all non-relevant points at once yields a training set inducing the same classification as the original training set. A series of algorithms have been developed to efficiently compute the set of relevant points. The current best algorithm due to Flores-Velazco [17] finds the set of relevant points in any fixed dimension in $O(nk^2)$, where k is the number of relevant points. This algorithm is a slightly adjusted version of the algorithm of Eppstein [15].

While no single point can be removed from the set of relevant points without changing the classification, it is possible that removing *multiple* relevant points at the same time does *not* change the classification. Therefore, the set of relevant points is not necessarily the smallest subset of the training set inducing the same classification. This is illustrated in Figure 1. In fact, it is not even guaranteed that the set of relevant points *contains* such a smallest subset. This is illustrated in Figure 2. In the paper introducing his algorithm to find the relevant points, Eppstein [15] conjectures that in high dimensions, finding such a smallest subset inducing the same classification is a much harder problem than finding the

*The full version of this paper can be retrieved from `http://arxiv.org/abs/2302.02132`

†Department of Mathematics, ETH Zurich, `rohrerj@student.ethz.ch`

‡Department of Computer Science, ETH Zurich, `simon.weber@inf.ethz.ch`

Simon Weber is supported by the Swiss National Science Foundation under project no. 204320.

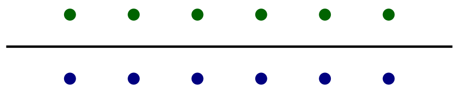


Figure 1: A training set in which all points are relevant, however, a single pair of vertically opposed points suffices to generate the same decision boundary (black).

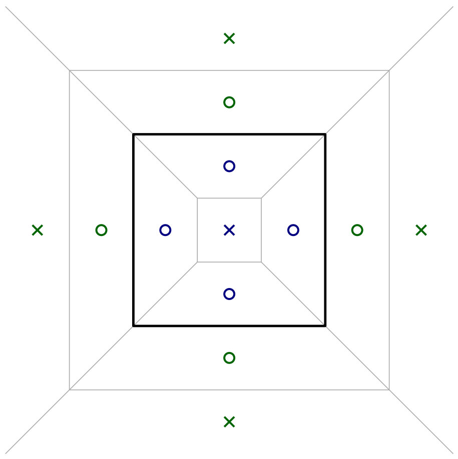


Figure 2: A training set for which the relevant points (circles) do not contain the smallest possible subset inducing the same classification (crosses). The decision boundary is drawn in black, while the Voronoi diagram of the training set is drawn in gray.

relevant points. In this paper, we show that high dimensions are not needed, and this problem is already NP-hard for binary classification in dimensions $d \geq 2$.

Note that lossless training set reduction is not the only studied method. We discuss alternative methods which are allowed to (slightly) change the induced classification later in Section 1.4.

1.1 Definitions

Definition 1 *A* labelled point set (m, P, c) *is given by an integer* m*, a set* $P \subset \mathbb{R}^d$ *of size* n*, and a classification function* $c : P \to [m]$*. We call* $c(p)$ *the* label *of* p*.*

We write $\mathrm{nn}(q, P)$ for the set of nearest neighbors of q in P. We say a point set is in *general position* if it contains no three collinear points and no four cocircular points. Note that by this definition, no point set $P \subset \mathbb{R}^1$ of at least three points is in general position.

Definition 2 *A labelled point set* (m, P, c) *induces the nearest neighbor classification* $f : \mathbb{R}^d \to 2^{[m]}$*, where*

$$f(q) = \{c(p) \mid p \in nn(q, P)\}.$$

Definition 3 *A* reduced training set *of some labelled point set* (m, P, c) *is a set* $Q \subseteq P$ *such that* $(m, Q, c|_Q)$ *induces the same nearest neighbor classification as* (m, P, c)*. A reduced training set of minimum cardinality among all reduced training sets is called a* minimum cardinality reduced training set*.*

Definition 4 *The decision problem* d-`MinNN` *is to decide if there exists a reduced training set of a given labelled point set* (m, P, c) *of at most* k *points.*

Definition 5 *A point* $p \in P$ *is a* relevant point *if* $(m, P \setminus \{p\}, c|_{P\setminus\{p\}})$ *and* (m, P, c) *induce different nearest neighbor classifications. We write* $\mathrm{rel}(P)$ *for the set of all relevant points.*

1.2 Results

We are now ready to state our results. As our first result, we show that in the case of training sets in general position, d-`MinNN` can be solved easily, since the relevant points already form a minimum cardinality reduced training set.

Theorem 1 *For an instance of* d-`MinNN` *where* P *is in general position,* $\mathrm{rel}(P)$ *is the unique minimum cardinality reduced training set.*

If this assumption of general position is not given, the set of relevant points is not guaranteed to be a minimum cardinality reduced training set. We show that generally, finding such a set is only feasible in dimension one, and NP-complete otherwise.

Theorem 2 1-`MinNN` *is in* P*.*

Theorem 3 *For any fixed dimension* $d \geq 2$*,* d-`MinNN` *is* NP*-complete, even when restricted to binary classification, i.e.,* $m = 2$*.*

1.3 Discussion

When data points are independently sampled from a probability distribution with (e.g., Gaussian) continuous noise, the resulting data set is in general position with probability 1. Theorem 1 thus implies that in practice, when a classification model is trained from high-precision data coming from a noisy source, computing the relevant points using the algorithms of Eppstein [15] or Flores-Velazco [17] is an efficient way to reduce the size of the training set to the optimum in a lossless fashion. The only way to reduce the size of the training set any further is to accept some small errors. While our results do not imply hardness of approximative training set reduction, Theorem 3 shows that it cannot be achieved efficiently by first "de-noising" the training set, and then finding the minimum cardinality reduced training set.

1.4 Related Work

Much of the work on nearest neighbor training set reduction has focused on detecting relevant points. Since the number of relevant points k is expected to be very small compared to the total number of points n, algorithms to find relevant points are ideally output-sensitive. The first such algorithm has been found by Clarkson in 1994 [11]. Bremner et al. [6] improved on Clarkson's algorithm for two-dimensional data with two labels. Recently, Eppstein [15] gave an algorithm for all dimensions and any number of labels, which is based on the simple geometric primitives of computing Euclidean minimum spanning trees and extreme points of point sets. Flores-Velazco [17] then showed that the Euclidean minimum spanning tree step can be skipped, yielding an $O(nk^2)$ algorithm for any constant dimension d.

Lossy reduction of nearest neighbor training sets, i.e., reduction in a way that slightly changes the classification, is often called *nearest neighbor condensation* in the literature. The most common concept in condensation is that of *consistent subsets*, introduced by Hart in 1968 [21]. A consistent subset is a subset of the training points that induces the same classification on the original training set, but not necessarily on all points of $\mathbb{R}^d$. It is known that computing a minimum cardinality consistent subset is NP-complete, for any number of labels $m \geq 2$ [26, 33]. *Selective subsets* [29] are subsets fulfilling a stronger condition than consistent subsets. Here, the distance from every point p in the original training set to a point with the same classification in the subset must be smaller than the distance from p to the nearest point of different classification in the original training set. Minimum cardinality selective subsets are also NP-complete to compute [34]. Flores-Velazco and Mount [18] introduced the approximative notions of α-consistency and α-selectivity and showed that it is NP-hard not only to find minimum cardinality α-consistent and α-selective subsets, but also to approximate their size beyond certain approximation factors. Due to all of these NP-hardness results, much of the recent research has focused on heuristic methods providing some guarantee on the resulting subset size [25].

So far we have only discussed training set reduction by taking a subset of the original data. Of course, another option is to construct a completely new training set that (approximately) induces the same classification as the original data while containing fewer points. Heath and Kasif [22] showed that an exact version of this approach is hopeless, since they show that finding the minimum number of points needed to create a Voronoi diagram containing a given polygonal tesselation as a substructure is NP-hard. This is a partial explanation to why this approach has not been studied much by the nearest neighbor community.

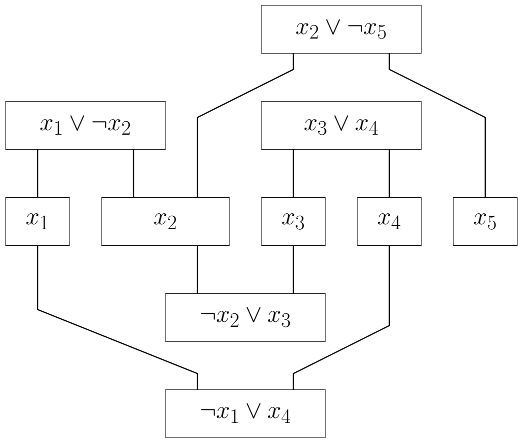


Figure 3: The type of embedding of the variable-clause graph used in the proof of Theorem 3.

1.5 Proof Techniques

The proof of Theorem 1 is very straightforward, and can be found in Appendix A. It makes use of the observation that for every reduced training set $Q \subseteq P$, every Voronoi wall of P between two regions of different classifications must lie in the bisecting hyperplane of some pair of points in Q. If P is in general position, no two pairs of points have the same bisecting hyperplane.

To prove Theorem 2 we provide a reduction from 1-MinNN to the problem of finding a maximum weight independent set on interval graphs, which is solvable in polynomial time [23].

Our proof of NP-hardness for Theorem 3 is similar to the proof of Heath and Kasif for the NP-hardness of the problem of finding Voronoi covers [22], which works by reduction from planar 3SAT. The proofs have two major differences. On one hand, a solution to d-MinNN must have the same classification on *all* of $\mathbb{R}^d$. In contrast, in the Voronoi cover problem, a fixed tesselation needs to appear only as a substructure in the Voronoi diagram. This means that we have to be more careful about introducing additional Voronoi walls in our training set. On the other hand, any solution to d-MinNN must be a subset of the training set, while in the Voronoi cover problem, arbitrary points are allowed. This gives us more control about the structure of possible solutions, and allows us to exclude unwanted solutions more easily.

Our proof works by reduction from the problem *V-cycle max2SAT*, a variant of max2SAT in which the bipartite variable-clause graph remains planar even after adding a Hamiltonian cycle through the vertices $x_1, \ldots, x_n$ corresponding to the variables. The planarity of this graph guarantees that we can efficiently find an embedding of the graph as shown in Figure 3. Then, every box corresponding to a variable x_i is replaced by a *variable gadget*. A variable gadget is a labelled point set that possesses two minimum cardinality reduced training sets. The choice between these two subsets indicates the value of the variable x_i. This value is then passed along the edges of the graph by *channels*. Finally, each box corresponding to a clause C_j is replaced by a *clause gadget*, a labelled point set for which the size of a minimum cardinality reduced training set is decreased by one if and only if at least one of two other points is already present. The size of a minimum cardinality reduced training set for the resulting labelled point set thus allows us to determine the largest number of simultaneously fulfillable clauses in the V-cycle max2SAT instance.

The main technical challenges in this reduction are

(i) avoiding unwanted interaction between gadgets, since points with different labels can interact over large distances in empty space, and

(ii) ensuring that the reduction yields a point set of only a polynomial number of points, with polynomial-sized coordinates.

2 The One-Dimensional Case

In this section, we provide a polynomial-time algorithm to find a minimum cardinality reduced training set in $\mathbb{R}^1$. In other words, we prove Theorem 2, 1-$\texttt{MinNN} \in \mathsf{P}$.

The classification induced by the given training set (m, P, c) decomposes $\mathbb{R}^1$ into a set $C = \{C_1, \ldots, C_t\}$ of open intervals of equal classification. We call the set $B = \{b_1, \ldots, b_{t-1}\}$ of points between these open intervals the *decision boundary points*. We begin with some observations holding for any reduced training set $(m, Q, c|_Q)$. First, for any $i \in [t]$, $Q \cap C_i \neq \emptyset$. Second, for any $i \in [t-1]$, b_i is the midpoint between its closest larger and smaller neighbor in Q. Finally, for any minimum cardinality reduced training set, we must have $Q \cap C_i \leq 2$.

Intuitively, towards a small reduced training set, we have to find a subset of P which often contains only one point per interval C_i, with this point being involved in defining both b_{i-1} and b_i. We formalize this in the following notion of a *chain*, as illustrated in Figure 4.

Definition 6 *A k-chain is a set $Q := \{q_1, \ldots, q_k\} \subseteq P$, for which there exists an integer i such that:*
(i) for any $j \in \{1, \ldots, k\}$, $q_j \in C_{i+j}$, and
(ii) for any $j \in \{1, \ldots, k-1\}$, $\frac{q_j+q_{j+1}}{2} = b_{i+j}$. We then say that Q covers *the boundary points $b_{i+1}, \ldots, b_{i+k-1}$.*

We say that two chains Q, Q' are *compatible*, if the intervals $[\min(Q), \max(Q)]$ and $[\min(Q'), \max(Q')]$ are disjoint. Compatible chains therefore cover disjoint sets of boundary points.

We now see that any minimum cardinality reduced training set must be the union of pairwise compatible chains. Any set of pairwise compatible chains can

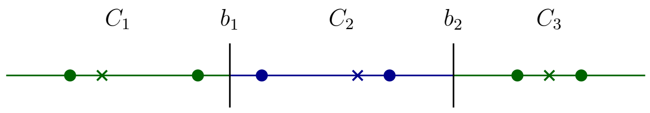


Figure 4: A labelled point set in $\mathbb{R}^1$ and a 3-chain (crosses) covering b_1 and b_2.

furthermore be completed to a reduced training set by adding 2-chains (consisting of relevant points) to cover the remaining uncovered boundary points.

If our reduced training set is a union of $k_1-, \ldots, k_\ell-$chains, the total number of points is $2(t-1) - \sum_{i=1}^{\ell}(k_i - 2)$, since every k_i-chain allows us to save a point in the $k_i - 2$ intervals between their first and last covered boundary points, compared to a naive solution with $2(t-1)$ points consisting only of 2-chains.

We are now ready to state our complete algorithm. First, we compute the set of boundary points and the set of all chains. Note that we can easily compute the set of all chains in $O(n^2)$, and that there are at most n^2 of them. Then, we associate each chain Q with the interval $[\min(Q), \max(Q)]$. Note now that a set of pairwise compatible chains is an independent set in the interval graph given by these intervals. We give each 2-chain a tiny weight $\epsilon > 0$, and each k-chain for $k > 2$ the weight $k - 2$. Finally, we use the dynamic programming approach of Hsiao, Tang and Chang [23] to find the *maximum weight independent set (MWIS)* within this graph. This algorithm is linear in the number of vertices, thus takes $O(n^2)$ in our case.

The resulting independent set corresponds to an inclusion-maximal independent set, with the maximum weight among all such sets. Its corresponding chains thus cover all boundary points, and their union is a minimum cardinality reduced training set.

3 NP-Completeness

In this section, we prove Theorem 3. We run the proof of NP-hardness for $d = 2$, since any instance of 2-$\texttt{MinNN}$ can be embedded in a 2-dimensional subspace of $\mathbb{R}^d$ to yield an instance of d-$\texttt{MinNN}$. The proof works by reduction from the following decision problem, which has been proven NP-hard by Buchin et al. [7][1].

Definition 7 *A conjunctive normal form formula $\phi = C_1 \wedge \ldots \wedge C_b$ over the variables $x_0, \ldots, x_{a-1}$ and an integer k form an instance of the* V-cycle max2SAT *problem, if every clause C_i is the disjunction ("or") of at most two literals and the graph $G_\phi = (V, E)$ is planar, where*

$$V = \{C_i \mid i \in [b]\} \cup \{x_i \mid 0 \leq i \leq a-1\},$$
$$E = \{(C_i, x_j) \mid x_j \in C_i\} \cup \{(x_i, x_{i+1 \bmod a}) \mid i \in [a]\}.$$

[1]A proof can be found in the appendix of the arXiv preprint [8].

The task is to decide whether there exists an assignment of the variables $x_0, \ldots, x_{a-1}$, such that at least k clauses of ϕ are fulfilled.

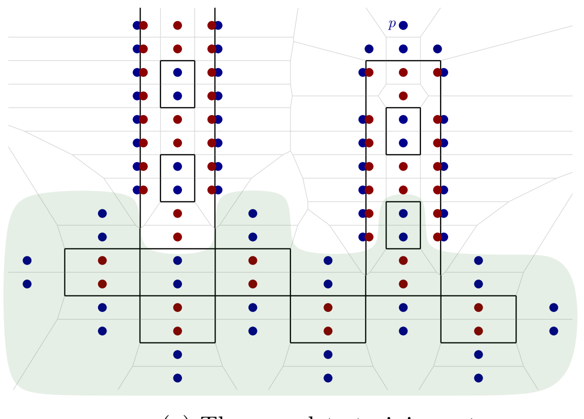
Note that the graph G'_ϕ obtained by replacing every clause vertex C_i (of degree 2) in G_ϕ by an edge is both planar and Hamiltonian, with the Hamiltonian cycle $(x_0, \ldots, x_{a-1}, x_0)$. Since every planar Hamiltonian graph has *book thickness* at most 2 [5], we can thus find an embedding of ϕ as in Figure 3 (for more details see the full version). Note that this embedding can be found in polynomial time. Given this embedding of ϕ, we will now construct a labelled point set $(2, P_\phi, c_\phi)$ encoding ϕ. For readability, we call the two labels "red" and "blue". The goal is that there is some number of points n', such that there exists a reduced training set with at most $n' - k$ points if and only if there exists an assignment of the variables in ϕ fulfilling at least k clauses.

To translate the embedding of ϕ into a training set, we show how to encode variables using *variable gadgets*, pass those variables along polylines using *channels*, and how to encode clauses using *clause gadgets*. To ensure that gadgets only interact with each other as intended, each gadget is designed to classify all points outside of some bounded area to be blue.

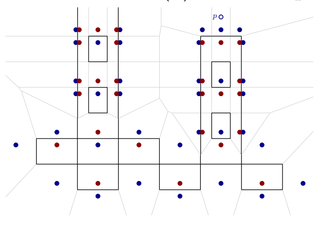
A *variable gadget* can be seen in Figure 5, where it is marked in green. The variable gadget consists of an upper and a lower *half*, separated by a contiguous horizontal part of the decision boundary. Each half is further split into *columns* of points. In both the upper and the lower half of this gadget, there are only two strict subsets which lead to the same classification: The outer points encoding "true" (Figure 5b), and the inner points encoding "false" (Figure 5c). Note that both of these subsets have the same number of points. The 6 uppermost and 6 lowermost blue points of the variable gadget are *shielding points* only used to ensure that every point outside of the gadget is classified blue.

Channels are infinitely extendable gadgets that can carry truth values. They can be attached to the top or bottom of a variable gadget to connect the variable gadget to clause gadgets above or below. See Figure 5 for an illustration of two channels attached to the top of the marked variable gadget. If a channel connects to a clause gadget in which the variable occurs positively, the channel is attached to a column *without* shielding points (left in Figure 5). Otherwise, if the variable occurs negatively, the channel is attached to a column *with* shielding points (right in Figure 5). The truth value carried by a channel can be read off by checking for the presence of the point p at the end of a channel.

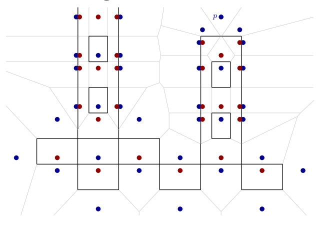
Channels are never attached to the outermost columns, and channels on the same half of a variable gadget leave at least two columns of space in between. Note that a variable gadget can be extended horizontally, allowing for an arbitrary number of channels.



(a) The complete training set.



(b) The "true" subset (not including p).



(c) The "false" subset (including p).

Figure 5: A variable gadget (green) encoding x with two channels attached on the top. The left channel leads to a clause containing x, and the right channel to one containing $\neg x$: p is only present when $\neg x$ is true.

The distances between points have been chosen very carefully. We say the vertical distance between two neighboring points in a variable gadget (and channel) is 1. The horizontal distance between points in a variable gadget is chosen to be 3.2. This is a very deliberate choice, since on the one hand, a too small horizontal distance would make the point p obsolete in Figure 5c. On the other hand, a too large horizontal distance would make some of the pairs of red and blue points that are used to generate the left and right vertical boundary of the channel obsolete. A distance of 3.2 avoids both of these issues simultaneously.

To allow us to connect all variable and clause gadgets, channels need some more flexibility. We can add *bends* of some fixed small angle to channels (see the full version of the paper). We can also *stretch* channels longitudinally (in the direction of their repeating pattern), by simply increasing the distance between a row with blue center point and a row with red center point. With the capability of creating bends and stretching longitudinally, we can make the point p of a channel lie at the location needed to attach it to a clause gadget.

A clause gadget is shown in Figure 6. It contains two special points, marked p_1 and p_2. These points are the point p of the two channels carrying the values of

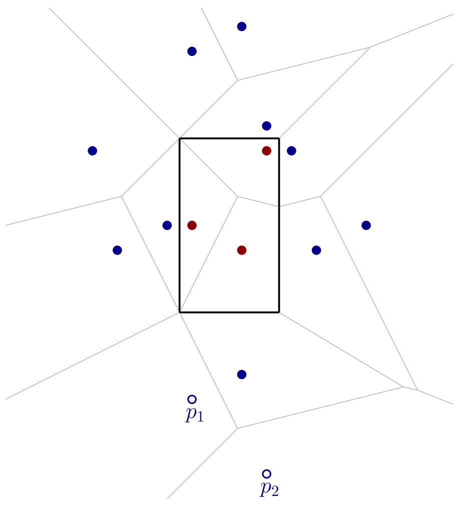


Figure 6: The complete training set of the clause gadget.

the involved variables to the clause gadget, respectively. Note that to be able to fit these channels without disturbing the function of the clause gadget, p_1 and p_2 need to have small vertical distance (say, 0.5) and large horizontal distance (say, 5). The clause gadget can also be drawn with these distances, but for legibility, the horizontal and vertical distances have been equalized in Figure 6. A figure showing the clause gadget with correct distances can be found in Appendix B.

If the value carried by the first (second) channel is true, the point p_1 (p_2) is part of the reduced training set, and otherwise it is not. The clause gadget is built in such a way that it needs 5 points if neither of p_1 and p_2 is present, and 4 points otherwise (see the full version for an illustration). Thus, we can save one point in the clause gadget if and only if the corresponding clause is fulfilled by the variable assignment corresponding to the points present in the channels.

We are now ready to prove Theorem 3.

Proof. For any fixed d, we can verify that a given subset of points is a reduced training set by computing the Voronoi diagrams of the training set and the subset and comparing the classifications. As the Voronoi diagram of n points in $\mathbb{R}^d$ can be computed in polynomial time [10], this proves NP-containment.

Towards proving NP-hardness, we reduce from V-cycle max2SAT, as in Definition 7. Given an instance (ϕ, k) we find an embedding of the bipartite variable-clause graph of ϕ as in Figure 3. Every box corresponding to a variable is replaced by a variable gadget, every box corresponding to a clause is replaced by a clause gadget, and the edges are replaced by channels (possibly with up to 2 bends and some number of stretchings). This point set can be constructed in polynomial time.

Let n_1 be the number of points in the reduced training set of all variable gadgets and channels corresponding to some fixed assignment of truth values to variables. Let n_2 be $5 \cdot b$ (recall that b is the number of clauses in ϕ). We will now prove that there exists a minimum cardinality reduced training set of size at most $n_1 + n_2 - k$ if and only if there exists an assignment of variables fulfilling at least k clauses of ϕ.

The "if" direction is trivial: If such an assignment of variables exists, it can clearly be translated into a reduced training set of the correct size, since each fulfilled clause gadget only requires four points.

For the "only if" direction, we argue that any reduced training set can be turned into a reduced training set corresponding to a variable assignment, without increasing the number of points. We first consider the channels. If any channel (including its endpoint p) is using any reduced subset other than the "true" or "false" subset (shown in Figure 5), for example if it is switching its value along the way, then it must be using strictly more points than if it was consistently encoding "true" or "false". We can then change the subset within the channel to be the subset that matches the truth value encoded within the half of the variable gadget the channel is attached to. Since this may remove the point p, this could cost us one additional point in the clause gadget, but we will also save at least one point within the channel. Next, we fix the variables. If any variable gadget is using any reduced subset other than the "true" or "false" subset, it is using all points on at least one of the two (upper and lower) halves, and in the other half it must be using either (a) all points, or (b) only the "false" subset. Let m be the number of columns in the gadget. In case (a), we can switch the gadget to using the "false" subset in both halves, and the attached channels and clauses are switched accordingly. This may require us to use one additional point per connected clause gadget, but saves at least $2m$ points in the variable gadget. Since not all columns of a variable gadget can be occupied by channels in both halves, this is a net improvement. In case (b), we switch the gadget to using the "true" subset in both halves. This again may require one additional point per clause gadget connected to the half that was previously false, but saves at least m points in the variable gadget, which is again a net improvement.

We can thus conclude that our training set has at least one minimum cardinality reduced training set which corresponds to a variable assignment, proving the correctness of our reduction. □

Acknowledgments. We thank David Eppstein for his great talk at SOSA'22 that inspired this work, Bernd Gärtner for his valuable advice, and the anonymous reviewers for their detailed feedback.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan 2008.

[2] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1), Nov 2009.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, Nov 1998.

[4] F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company, 2013.

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

[6] D. Bremner, E. Demaine, J. Erickson, J. Iacono, S. Langerman, P. Morin, and G. Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. *Discrete & Computational Geometry*, 33(4):593–604, 2005.

[7] K. Buchin, V. Polishchuk, L. Sedov, V. Roman, et al. Geometric secluded paths and planar satisfiability. In *36th International Symposium on Computational Geometry (SoCG 2020), June 22-26, 2020*, volume 164, pages 24–1, 2020.

[8] K. Buchin, V. Polishchuk, L. Sedov, and R. Voronov. Geometric secluded paths and planar satisfiability. *arXiv preprint arXiv:1902.06471*, 2019.

[9] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20(3):359–373, Oct 1998.

[10] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(4):377–409, 1993.

[11] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 695–702. IEEE, 1994.

[12] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[13] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, page 253–262, New York, NY, USA, 2004. Association for Computing Machinery.

[14] R. A. Dwyer. Higher-dimensional voronoi diagrams in linear expected time. *Discrete & Computational Geometry*, 6(3):343–367, Sep 1991.

[15] D. Eppstein. Finding relevant points for nearest-neighbor classification. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 68–78. SIAM, 2022.

[16] D. Eppstein, M. T. Goodrich, and J. Z. Sun. Skip quadtrees: Dynamic data structures for multidimensional point sets. *International Journal of Computational Geometry & Applications*, 18(01n02):131–160, 2008.

[17] A. Flores-Velazco. Improved Search of Relevant Points for Nearest-Neighbor Classification. In S. Chechik, G. Navarro, E. Rotenberg, and G. Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:10, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[18] A. Flores-Velazco and D. M. Mount. Coresets for the nearest-neighbor rule. In F. Grandoni, G. Herman, and P. Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 47:1–47:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[19] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *25th VLDB Conference*, pages 518–529, 1999.

[20] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of delaunay and voronoi diagrams. *Algorithmica*, 7(1):381–413, Jun 1992.

[21] P. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[22] D. Heath and S. Kasif. The complexity of finding minimal voronoi covers with applications to machine learning. *Computational Geometry*, 3(5):289–305, 1993.

[23] J. Y. Hsiao, C. Y. Tang, and R. S. Chang. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. *Information Processing Letters*, 43(5):229–235, 1992.

[24] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery.

[25] N. Jankowski and M. Grochowski. Comparison of instances seletion algorithms i. algorithms survey. In L. Rutkowski, J. H. Siekmann, R. Tadeusiewicz, and L. A. Zadeh, editors, *Artificial Intelligence and Soft Computing - ICAISC 2004*, pages 598–603, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[26] K. Khodamoradi, R. Krishnamurti, and B. Roy. Consistent subset problem with two labels. In B. Panda and P. P. Goswami, editors, *Algorithms and Discrete Applied Mathematics*, pages 131–142, Cham, 2018. Springer International Publishing.

[27] T. Liu, A. Moore, K. Yang, and A. Gray. An investigation of practical approximate nearest neighbor algorithms. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.

[28] F. P. Preparata and R. Tamassia. Efficient point location in a convex spatial cell-complex. *SIAM Journal on Computing*, 21(2):267–280, 1992.

[29] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.

[30] R. Seidel. Exact upper bounds for the number of faces in d-dimensional voronoi diagrams. In *Applied Geometry And Discrete Mathematics*, 1990.

[31] Z. Usiskin. *The classification of quadrilaterals: A study in definition*. Information Age Publishing, 2008.

[32] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*. *The Computer Journal*, 24(2):167–172, Jan 1981.

[33] G. Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, page 224–233, New York, NY, USA, 1991. Association for Computing Machinery.

[34] A. V. Zukhba. NP-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recognit. Image Anal.*, 20(4):484–494, Dec 2010.

A Proof of Theorem 1

In this section we wish to prove Theorem 1. Since we already know that for a labelled point set (m, P, c) the set $\text{rel}(P)$ of relevant points is a reduced training set, it suffices to show that any reduced training set must include all relevant points.

Let us first introduce a few definitions. A *Voronoi wall* is a cell of the Voronoi diagram of dimension $d-1$. These walls separate two fully-dimensional Voronoi cells. A Voronoi wall that separates two cells in which the nearest neighbor rule induced by P gives a different classification is said to be part of the *decision boundary*.

Observation 1 *For any reduced training set given by $Q \subseteq P$, a Voronoi wall of P which is part of the decision boundary must be (a subset of) some Voronoi wall of Q as well, since Q must induce the same nearest neighbor classification as P.*

Note that every Voronoi wall W is a subset of the bisecting hyperplane of the two points belonging to the incident Voronoi cells. Since W is $(d-1)$-dimensional, it uniquely determines this hyperplane. We next show that under the general position assumption, every hyperplane can be the bisecting hyperplane of at most one pair of points.

Lemma 4 *For any point set $P \subset \mathbb{R}^d$ in general position, no two distinct pairs of points in P have the same bisecting hyperplane.*

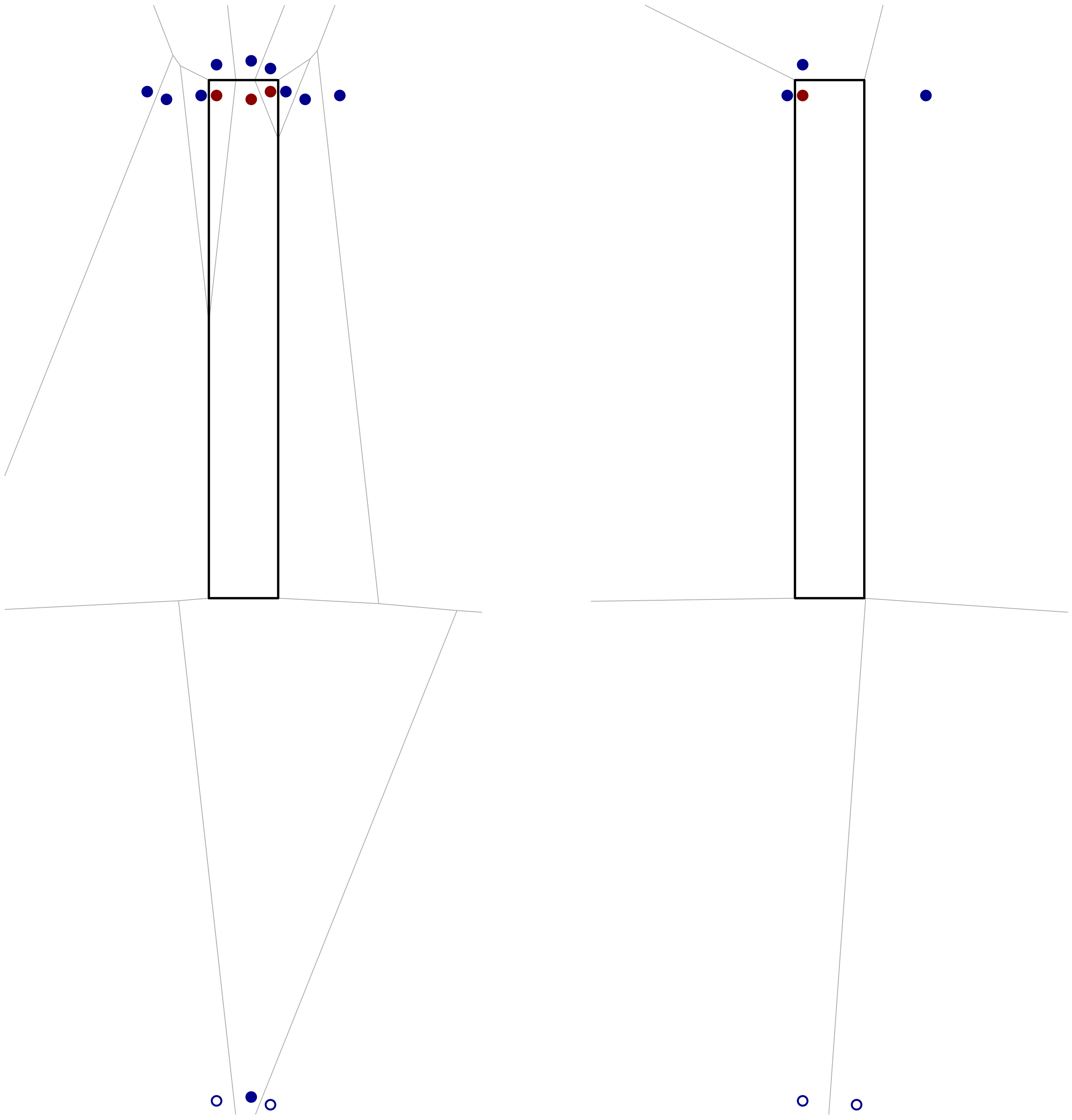
Proof. Towards a contradiction, assume $a, b \in P$ and $c, d \in P$ have the same bisecting hyperplane. First note that a, b, c, d must all be distinct, since a, b and a, d have different bisecting hyperplanes if $b \neq d$.

The points a, b, c, d cannot be collinear, since general position assumption requires that no three points are collinear. Since a, b and c, d have the same bisecting hyperplane, the lines ab and cd must be parallel. Thus, a, b, c, d must lie on a common plane. Furthermore, they must be the corners of an isoceles trapezoid, a *cyclic quadrilateral* [31]. Thus, a, b, c, d are cocircular, forming a contradiction with the general position assumption. $\square$

Every relevant point $p \in \text{rel}(P)$ shares a Voronoi wall with some point q with a label $c(q) \neq c(p)$, i.e., a wall that is part of the decision boundary: Otherwise, removing p would not change the classification, since removing a point p from a Voronoi diagram distributes the interior of the Voronoi cell of p to the Voronoi cells of the points q such that p and q previously shared a Voronoi wall. By Observation 1 and Lemma 4, we know that p, q must therefore be part of every reduced training set. Since this holds for every relevant point p, any reduced training set given by Q must contain the set $\text{rel}(P)$, and Theorem 1 follows.

B Clause Gadget

Below, in Figure 7, the clause gadget is shown to scale, with the distance between the two attachment point for channels p_1 and p_2 having horizontal distance of 5 and vertical distance of 0.5. The vertical stretching of the gadget is necessary to ensure that the classification remains correct in the case where both variables are true (Figure 7b). If the gadget would be less stretched, a Voronoi wall between the red point and p_2 would appear.



(a) The complete training set.

(b) Both variables are true.

Figure 7: The clause gadget with the correct distances between the attachment points.

Clustering with Neighborhoods is Hard for Squares[*]

Georgiy Klimenko[†] Benjamin Raichel[‡]

Abstract

In the clustering with neighborhoods problem one is given a set $\mathcal{S}$ of disjoint convex objects in the plane and an integer parameter $k \geq 0$, and the goal is to select a set C of k center points in the plane so as to minimize the maximum distance of an object in $\mathcal{S}$ to its nearest center in C. Previously [HKR21] showed that this problem cannot be approximated within any factor when $\mathcal{S}$ is a set of disjoint line segments, however, when $\mathcal{S}$ is a set of disjoint disks there is a roughly 8.46 approximation algorithm and a roughly 6.99 approximation lower bound. In this paper we investigate this significant discrepancy in hardness between these shapes. Specifically, we show that when $\mathcal{S}$ is a set of axis aligned squares of the same size, the problem again is hard to approximate within any factor. This surprising fact shows that the discrepancy is not due to the fatness of the object class, as one might otherwise naturally suspect.

1 Introduction

Given a set of n points P in a metric space and an integer parameter $k \geq 0$, in the standard k-center clustering problem the goal is to select a set C of k center points from the metric space (or in the discrete variant $C \subseteq P$) so as to minimize the maximum distance of a point in P from its nearest center in C. This fundamental problem and its variations have been well studied in the computational geometry community. For the standard problem there is a well known greedy 2-approximation algorithm due to Gonzalez [Gon85], and an iterative scooping based 2-approximation algorithm due to Hochbaum and Shmoys [HS85]. Conversely, for general metric spaces it is NP-hard to approximate within any factor less than 2, and even for points in the plane the problem remains hard to approximate within a factor of roughly 1.82 [FG88].

While many variants of k-center clustering have been considered, here we focus on the problem of *k-center clustering with neighborhoods* introduced recently in [HKR21]. In this problem the input is a set $\mathcal{S}$ of n disjoint convex objects in the plane, and the goal is again to select a set C of k points from the plane so as to minimize the maximum distance of an object in $\mathcal{S}$ from its nearest center in C. Note that the standard k-center problem is a special case of k-center clustering with neighborhoods where $\mathcal{S} = P$ is a discrete point set.

In [HKR21] it was shown that clustering with neighborhoods is hard to approximate within any factor when $\mathcal{S}$ is a set of disjoint segments. Conversely, it was also shown that when $\mathcal{S}$ is a set of disjoint disks the problem is $\frac{\sqrt{13}-\sqrt{3}}{2-\sqrt{3}} \approx 6.99$ hard to approximate, and additionally a near matching $(5+2\sqrt{3}) \approx 8.46$ approximation algorithm was given. In other words, for disks the problem is APX-complete.

The clustering with neighborhoods problem can be equivalently defined as finding k equal radius balls of the smallest possible radius such that every object has non-empty intersection with at least one of the balls. Alternatively, one could require that each object is entirely contained in one of the balls. This however, implies that the optimal radius is at least the radius of the largest object, whereas in our case the optimal radius can be arbitrarily smaller. This significantly and provably changes the hardness of the two problems. Specifically, Xu and Xu [XX10] considered k-center clustering on point sets where given points sets $S_1, \ldots, S_n$ the goal is to find k balls of minimum radius such that each S_i is entirely contained in one of the balls. For this problem they achieved a $(1+\sqrt{3})$-approximation, whereas clustering with neighborhoods cannot in general be approximated within any factor in polynomial time unless $\mathsf{P} = \mathsf{NP}$.

Motivation and Contribution. As discussed above, clustering with neighborhoods is hard to approximate within any factor when the objects are disjoint line segments, however, when the objects are disks there is a constant factor approximation. This intriguingly large hardness gap between segments and disks begs the question, what geometric feature accounts for this gap? One may naturally suspect (as the authors did), that the difference is due to fatness, as segments are arbitrarily skinny objects whereas disks are fat. It is well known

[*]A preliminary version of the main proof in this paper appeared in Georgiy Klimenko's thesis [Kli23]

[†]Department of Computer Science; University of Texas at Dallas; Richardson, TX 75080, USA; gik140030@utdallas.edu; Work on this paper was partially supported by a NSF CAREER Award 1750780.

[‡]Department of Computer Science; University of Texas at Dallas; Richardson, TX 75080, USA; benjamin.raichel@utdallas.edu; http://utdallas.edu/~benjamin.raichel. Work on this paper was partially supported by a NSF CAREER Award 1750780.

that this basic geometric property can often make a significant difference in the difficulty of a problem (e.g. [Cha03]). Surprisingly, however, in this paper we show that when the objects are disjoint squares (one of the simplest classes of fat objects), not only does the constant factor approximation algorithm break down, but in fact the problem is again hard to approximate within any factor, as was the case for line segments. Moreover, we show this is true even when the squares are axis aligned and all of equal size. Indeed, this paper shows the hardness gap is not due fatness, but rather roughly speaking concerns more how pointed the objects are. (More precisely, it concerns how closely one can place three disjoint objects to a single point.)

2 Preliminaries

Given points $x, y \in \mathbb{R}^d$, $||x-y||$ denotes their Euclidean distance. Given two closed sets $X, Y \subset \mathbb{R}^d$, $||X-Y|| = \min_{x\in X, y\in Y} ||x-y||$ denotes their distance. For a point x and a value $r \geq 0$, let $B(x,r)$ denote the closed ball centered at x and with radius r. [HKR21] considered the following problem.

Problem 1 (Clustering with Neighborhoods) *Given a set $\mathcal{S}$ of n disjoint convex objects in the plane, and an integer parameter $k \geq 0$, find a set of k points C (called centers) which minimize the maximum distance to a convex object in $\mathcal{S}$. That is,*

$$C = \arg \min_{C' \subset \mathbb{R}^2, |C'|=k} \max_{S\in\mathcal{S}} ||S - C'||.$$

Let C be any set of k points, and let $r = \max_{S\in\mathcal{S}} ||S - C||$. We refer to r as the *radius* of the solution C, since r is the minimum radius such that the set of all balls $B(c,r)$ for $c \in C$, intersect all $S \in \mathcal{S}$. If C is an optimal solution then we refer to its radius r_{opt} as the optimal radius. Let $\mathcal{S}, k$ be an instance of Problem 1 with optimal radius r_{opt}. For a value $\alpha \geq 1$, we refer to a polynomial time algorithm as an α-approximation algorithm if it returns a solution C of size k such that the radius is $\leq \alpha r_{opt}$.

3 Hardness for Squares

In this section we argue that it is hard to approximate Problem 1 within any factor when $\mathcal{S}$ is a set of axis aligned squares of the same size. Our hardness results use a construction similar to the one from [FG88], where they reduce from the problem of planar vertex cover where the maximum degree of a vertex is three. This problem is known to be NP-complete [GJ77], and we denote this problem as P3VC. We remark that the high level approach of reducing from P3VC used in [FG88] has inspired many other hardness reduction for geometric problems, including the prior reductions for clustering with neighborhoods for the cases of segments and disks [HKR21].

Let G, k be an instance of P3VC, and consider a straight line embedding of G. In particular, in $O(n \log n)$ time one can compute an straight line embedding of G where the vertices are on a $2n-4 \times n-2$ grid [FPP90]. We now scale this graph by a polynomial factor large enough to ensure two properties. First, for every edge of G there is a portion of that edge which has length at least say 100 and the closest other edge or vertex is distance at least 100. Call this the *free zone* of the edge. Second, for each vertex of G, there is a ball centered at that vertex, such that this ball only intersects the at most 3 adjacent edges of that vertex, does not intersect the free zones of those edges, and the intersection points of the edges with the boundary of the ball are at least distance 10 apart from one another. Call this the *free zone* of the vertex. Note ensuring these two properties only requires scaling by a polynomial factor since the graph was initially embedded on a roughly $n \times n$ grid.

We now describe how to replace each edge of G with a sequence of unit squares. Roughly speaking this sequence of unit squares will be the unit grid cells that the edge overlaps, i.e. the standard pixelized representation of the edge. However, there will be several key differences, particularly inside the free zone of each edge and vertex, which we describe below. Outside the free zones we will simply include the unit squares of the grid cells intersected by the edge, except when the edge intersects 3 of the 4 grid cells adjacent to a grid point. In this case will will only include the diagonally adjacent squares. See Figure 3.1. Note in general there may be

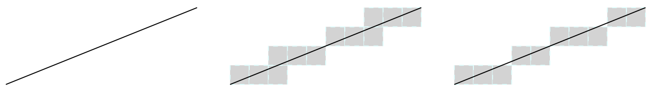


Figure 3.1: Left: A portion of an edge. Middle: Grid cells intersected. Right: When three cells adjacent to a grid point are intersected, only the diagonally adjacent pair is kept.

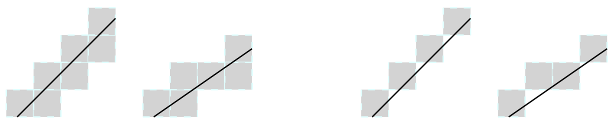


Figure 3.2: Left: Two cases with consecutive grid points where 3 adjacent grid cells are intersected. Right: Iteratively removing intersected grid cells so that no grid point is adjacent to more than 2.

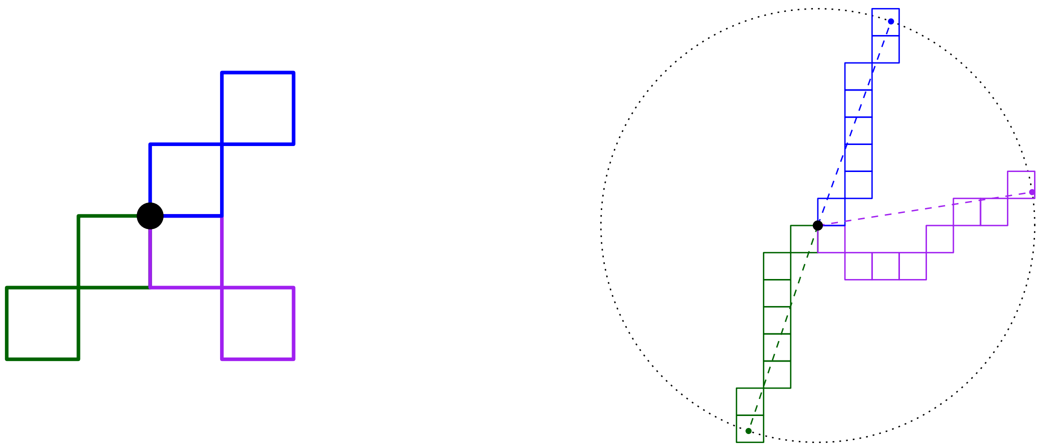


Figure 3.3: Left: The first two squares of each of the three edges adjacent to a vertex. Right: A routing of the square sequence for each adjacent edge which leads to the intersection point of that edge with the ball.

multiple grid points in a row where 3 of the 4 adjacent grid cell squares are intersected. Thus to be more precise, going from left to right, we iteratively remove the third adjacent grid cell square (again the square that is not in the diagonal pair), until all grid points are adjacent to at most 2 remaining intersected squares. See Figure 3.2.

Now we describe the construction within the free zone of a given vertex v, with adjacent edges e_1, e_2, e_3. (If v has fewer adjacent edges the construction is only simpler.) Within the free zone of v we cannot simply pixelize the edges as described above, since the angle of the adjacent edges might be such that say e_1 and e_2 initially pass through the same neighboring grid cell of v. Instead we enforce that the square sequence for each edge start on a distinct square adjacent to v, and moreover, the second square in the sequence for each edge continue in this diagonal direction from v. (This condition on the second square in the sequence of the edge ensures that squares from different edges are only adjacent at v itself.) See Figure 3.3. As described above when defining the free zone of v there exists a ball centered at v such that the points of intersection of e_1, e_2, e_3 with the boundary of this ball are at least distance 10 apart from one another. This ample spacing ensures that we can route the sequences of squares we are constructing for each edge such that squares from different edges stay at least distance 1 apart from each other (except at v itself) and that the square sequence for each edge ends up on its respective intersection point on the ball boundary. See Figure 3.3.

Finally, the last part of the construction concerns the free zone of each edge. So consider a given edge e. If e consists of a sequence of an odd number of squares after applying the above pixelization process to e along with the above modifications in the vertex free zones of its endpoints, then we leave the free zone of e untouched (i.e. it is just pixelized like the rest of e). However, if e consists of a sequence of an even number of squares then in the free zone we make the following modification so that the total number of squares is odd. We consider two cases. First, if within the free zone the sequence of squares has at least 6 consecutive squares which are horizontally adjacent (or 6 which are vertically adjacent), then we replace these 6 squares with the parity shifting gadget show in Figure 3.4 (where if the 6 squares where vertically adjacent we rotate the gadget 90 degrees).

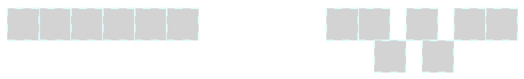


Figure 3.4: Left: 6 horizontally adjacent squares. Right: Parity gadget replacing the 6 squares with 7 squares. The horizontal gaps between squares on the top and bottom rows have length exactly 1/2.

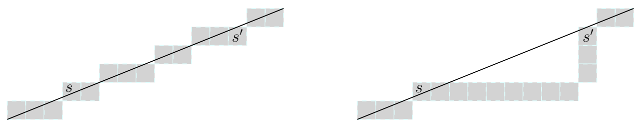


Figure 3.5: Left: Pixelized edge. Right: Replacing the portion of the pixelized edge between s and s' with an L shaped sequence of squares. Note the figure is not to scale, as the portion replaced did not have at least 12 squares.

Otherwise, if there are not 6 horizontally adjacent squares then we replace a portion of the square sequence in the free zone with an L shaped sequence, see Figure 3.5. In particular, viewing the squares in the sequence as ordered from left to right, pick a square s whose previous adjacent square is diagonally adjacent. Next pick a square s' which is at least 12 squares after

s in the sequence and such that the square after s' is diagonally adjacent. Now replace all squares between s and s' with an L shaped sequence consisting of single run of horizontally adjacent squares followed by a run of vertically adjacent squares (again see Figure 3.5). Now if this new sequence of squares between s and s' has a different parity than the original sequence between s and s' then we are done. Otherwise, either the horizontal or vertical portion of this L shaped sequence must have at least 6 squares and thus we can insert the same parity gadget described above into this portion of the L shaped sequence.

Let $0 < \varepsilon \ll 1/4$ be some value. For the final step in our construction, we now shrink all of the above created squares (about their respective centerpoints) such that two squares diagonally adjacent to the same grid point are distance 2ε apart from one another. (Note this means horizontally or vertically adjacent squares are $2\varepsilon/\sqrt{2}$ apart.)

So given an instance G, k of P3VC, we construct an instance $\mathcal{S}, \kappa$ of Problem 1 where $\mathcal{S}$ is determined from G as described above and $\kappa = k + (|\mathcal{S}| - |E|)/2$. We first argue if G has a vertex cover of size k then for our instance of Problem 1 there is a solution of radius ε. First, for any vertex v in the vertex cover we create a center, and place it at the location of v in the embedding. By the way we shrunk the squares, $B(v, \varepsilon)$ will intersect the (at most) three adjacent initial squares of v's adjacent edge sequences. We now cover the remaining squares with $(|\mathcal{S}| - |E|)/2$ centers. For any edge $e \in E$ let n_e be the number of squares used for e in the above construction. Observe that as we already placed centers at vertices corresponding to a vertex cover of the edges, at least one square of each edge is already covered, and so there are at most $n_e - 1$ consecutive squares that need to be covered. (Note $n_e - 1$ is even.) However, as consecutive squares are at most 2ε apart on each edge, these $n_e - 1$ squares can be covered with $(n_e - 1)/2$ balls of radius ε by covering the squares in pairs. Thus the total number of centers used is $k + \sum_{e \in E}(n_e - 1)/2 = k + (|\mathcal{S}| - |E|)/2 = \kappa$.

Now suppose the minimum vertex cover of G requires $> k$ vertices. In this case we argue that our instance of Problem 1 requires more than κ centers if we limit to balls with radius $< 1/4$. Call any two squares in $\mathcal{S}$ neighboring if they are consecutive on an edge or if they are squares on the v end of two edges adjacent to a vertex v. By construction, neighboring squares have distance $\leq 2\varepsilon$ from each other. For a pair of squares which are not neighboring their distance is at least $1/2$. Specifically, within the free zone of a vertex we ensured squares from different edges were at least unit distance apart (except at the vertex). Also, squares from differing edges remain at least unit distance apart outside of the free zones of vertices. For two squares from the same edge, the pixelization process enforces at least unit distance for non-adjacent squares. The same holds for inserting L shaped sequences in the free zone of an edge. Thus all that remains is the parity gadget, where the closest two non-adjacent squares can be is exactly $1/2$.

By the above, limiting to radius $< 1/4$ therefore implies that, other than at the (up to) three neighboring squares at a vertex, any ball either covers just a single square, or a pair of neighboring squares. An edge e with n_e squares thus requires at least $\lceil n_e/2 \rceil = 1 + (n_e - 1)/2$ balls to cover it. Moreover, a ball can only cover both a square of e and e' if those squares are on the v end of two edges adjacent to v. Let E_z be the subset of edges with at least one square covered by such a ball (i.e. a ball corresponding to a vertex), and let z be the number of such balls. Then the total number of balls required is

$$\geq z + \sum_{e \in E_z} (n_e - 1)/2 + \sum_{e \in E \setminus E_z} (1 + (n_e - 1)/2)$$
$$= z + (|\mathcal{S}| - |E|)/2 + |E \setminus E_z| = z + (\kappa - k) + |E \setminus E_z|,$$

which is more than κ when $z + |E \setminus E_z| > k$. Notice, however, there is a vertex cover of G of size $z + |E \setminus E_z|$, consisting of the vertices that z counted, and one vertex from either end of each edge in $E \setminus E_z$. Thus as the minimum vertex cover has size $> k$, we have $z + |E \setminus E_z| > k$ as desired.

Therefore, if we could approximate the minimum radius of our Problem 1 instance within any factor less than $\frac{1/4}{\varepsilon} = \frac{1}{4\varepsilon}$ then we can determine whether the corresponding vertex cover instance had a solution with $\leq k$ vertices. However, we are free to make $\varepsilon > 0$ as small we want, and thus $\frac{1}{4\varepsilon}$ as large as we want, so long as this quantity (or more precisely a lower bound on it) is computable in polynomial time. Thus we have the following theorem.

Theorem 2 *Problem 1 cannot be approximated within any factor in polynomial time unless* P = NP, *even when restricting to the set of instances in which $\mathcal{S}$ is a set of axis aligned squares of the same size.*

References

[Cha03] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003.

[FG88] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444. ACM, 1988.

[FPP90] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Comb.*, 10(1):41–51, 1990.

[GJ77] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[Gon85] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[HKR21] H. Huang, G. Klimenko, and B. Raichel. Clustering with neighborhoods. In *32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212 of *LIPIcs*, pages 6:1–6:17, 2021.

[HS85] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[Kli23] G. Klimenko. *Convex hull simplification and geometric hardness*. The University of Texas at Dallas, 2023.

[XX10] G. Xu and J. Xu. Efficient approximation algorithms for clustering point-sets. *Computational Geometry*, 43(1):59–66, 2010.

CCOSKEG Discs in Simple Polygons*

Prosenjit Bose† Anthony D'Angelo‡ Stephane Durocher§

Abstract

We consider the problem of finding a geodesic disc D of smallest radius containing at least k points among n inside a simple polygon P. The centre of D must lie on a chord in P. The polygon P has m vertices, r of which are reflex. We present an exact algorithm using parametric search that runs in $O(n \log^2 n+m)$ time with high probability and $O(n \log r + m)$ space.

1 Introduction

The *smallest / minimum enclosing disc* problem takes as input a set S of n points in the plane and returns the smallest Euclidean disc that contains S. This can be solved in $O(n)$ expected time [58] and $O(n)$ worst-case time [43]. The *smallest k-enclosing disc* problem is a generalization that asks for a smallest disc that contains at least $k \leq |S|$ points[1] of S, for any given k, and has been well studied [3, 24, 26, 27, 32, 39, 40]. It is conjectured that an exact algorithm that computes the smallest k-enclosing disc in the plane requires $\Omega(nk)$ time [31, §1.5].

Matoušek [39] presented an algorithm that first computes a constant-factor approximation[2] in $O(n \log n)$ time and $O(n)$ space (recently improved to $O(n)$ expected time for a 2-approximation that uses $O(n)$ expected space [32]), and then uses that approximation to seed an algorithm for solving the problem exactly in $O(n \log n + nk)$ expected time using $O(nk)$ space or $O(n \log n+nk \log k)$ expected time using $O(n)$ space (recently improved to $O(nk)$ expected time using $O(n+k^2)$ expected space [24, 32]). Matoušek [40] also presented an algorithm for computing the smallest disc that contains all but at most q of n points in $O(n \log n + q^3 n^\epsilon)$ time, where ϵ is "a positive constant that can be made arbitrarily small by adjusting the parameters of the algorithms; multiplicative constants in the $O()$ notation may depend on ϵ" [40].

In this paper we generalize the smallest k-enclosing disc problem to simple polygons using the *geodesic metric*, meaning that the distance $d_g(a, b)$ between two points a and b is the length of the shortest path $\Pi(a, b)$ between a and b that lies completely inside the simple polygon P. A *geodesic disc* $D(c, \rho)$ of radius ρ centred at $c \in P$ is the set of all points in P whose geodesic distance to c is at most ρ. Our article focuses on the *Chord-Constrained Smallest k-Enclosing Geodesic* (CCOSKEG) disc problem.

CCOSKEG Disc Problem

Consider a simple polygon P_{in} defined by a sequence of m vertices in $\mathbb{R}^2$, $r > 0$ of which are reflex vertices, a set S of n points of $\mathbb{R}^2$ contained in P_{in},[3] an integer $k \leq n$, and an input chord $\ell \subset P_{in}$.[4] Find a *CCOSKEG disc*, i.e., a geodesic disc of minimum radius ρ^* in P_{in} *centred on* ℓ that contains at least k points of S.

Without loss of generality, we consider ℓ to be the x-axis. We make the general position assumptions that no two points of S are equidistant to a vertex of P_{in}, and no four points of S are geodesically co-circular. Under these assumptions, a smallest k-enclosing geodesic (SKEG) or CCOSKEG disc contains exactly k points. Let $D(c^*, \rho^*)$ be a CCOSKEG disc for the points of S in P_{in} constrained to the input chord ℓ. For convenience, at times we will refer to this as simply D^*. A k-enclosing geodesic disc (*KEG* disc) is a geodesic disc in P_{in} that contains exactly k points of S. The main result of our article is the following theorem.

Theorem 4 *Given a chord $\ell \subset P_{in}$ we compute a CCOSKEG disc $D(c^*, \rho^*)$ in $O(n \log^2 n + m)$ time with high probability[5] using $O(n \log r + m)$ space.*

1.1 Related Work

Other than our work on SKEG discs [15], we are not aware of other work tackling the subject of this paper. In our previous work [15], we presented an algorithm to compute a 2-approximation SKEG disc that

*This work is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†Carleton University, Ottawa, Canada, jit@scs.carleton.ca

‡anthony.dangelo@carleton.ca

§University of Manitoba, Winnipeg, Canada, stephane.durocher@umanitoba.ca

[1]In this paper, we use the notation $|Z|$ to denote the number of points in Z if Z is a point set, or the number of vertices of Z if Z is a face or a polygon.

[2]An α-approximation means that the disc returned has a radius at most α times the radius of an optimal solution.

[3]When we refer to a point p being in a polygon P, we mean that p is in the interior of P or on the boundary, ∂P.

[4]We use the terms *chord* and *diagonal* interchangeably.

[5]We say an event happens with high probability if the probability is at least $1 - n^{-\lambda}$ for some constant λ.

runs in expected time $O(n \log^2 n \log r + m)$ and expected space $O(n + m)$ if $k \in O(n/\log n)$; if $k \in \omega(n/\log n)$, it computes such a disc with high probability in $O(n \log^2 n \log r + m)$ deterministic time with $O(n+m)$ space. We compared it to the approach we presented in the same paper that uses higher-order geodesic Voronoi diagrams to find the exact solution. Assuming general position, a SKEG disc has either two or three points of S on its boundary, allowing techniques involving Voronoi diagrams to be applied. Ignoring polylogarithmic factors, the worst-case runtime for the Voronoi diagram approach for $k = n$ is $O(n + m)$; for $k = n - 1$ and $r/\log^2 r \in \Omega(k \log k)$ is $O(nr+m)$; for $k = n-1$ and $r/\log^2 r \in o(k \log k)$ is $O(n^2+nr+r^2+m)$; for $k < n-1$ and for $n \log n \in o(r/\log r)$ is $O(k^2 n + \min(rk, r(n-k)) + m)$; and $O(k^2 n + k^2 r + \min(kr, r(n-k)) + m)$ otherwise. Higher-order Voronoi diagrams have been considered to solve the smallest k-enclosing disc problem in the plane [3, 26].

There has been other work done with geodesic discs in polygons. A region Q is *geodesically convex* relative to a polygon P if for all points $u, v \in Q$, the geodesic shortest path from u to v in P is in Q. The *geodesic convex hull* CH_g of a set of points S in a polygon P is the intersection of all geodesically convex regions in P that contain S. The geodesic convex hull of n points in a simple m-gon can be computed in $O(n \log n+m)$ time using $O(n + m)$ space [29, 53].

The *geodesic centre* problem asks for a smallest geodesic disc that lies in the polygon and encloses all vertices of the polygon (stated another way, a point that minimizes the geodesic distance to the farthest point). This problem is well studied [4, 11, 16, 48, 53] and can be solved in $O(m)$ time and space [4]. The geodesic centre problem has been generalized to finding the geodesic centre of a set of points S inside a simple polygon in $O(n \log n + m)$ time [10]. Generalized versions of the geodesic centre for polygons [12, 46, 47, 55, 56]; packing and covering [49, 55]; and clustering [14] have all been studied.

Dynamic k-nearest neighbour queries were studied by de Berg and Staals [25]. They presented a static data structure for geodesic k-nearest neighbour queries for n sites in a simple m-gon that is built in $O(n(\log n \log^2 m+\log^3 m))$ expected time using $O(n \log n \log m+m)$ expected space and answers queries in $O(\log(n + m) \log m + k \log m)$ expected time.

If P_{in} has no reflex vertices, it is a convex polygon and the SKEG disc problem is solved by the algorithm for planar instances which uses a grid-refinement strategy. This works in the plane because $\mathbb{R}^2$ with the Euclidean metric is a doubling metric space, meaning that for any disc of radius $\rho > 0$ in $\mathbb{R}^2$ it can be covered by $O(1)$ discs of radius $\rho/2$ [33]. Geodesic discs do not have this property; it may take $\Theta(r)$ smaller discs to cover the larger one (refer to Fig. 1 in Appendix B). Another difficulty of the geodesic metric is that for two points u and v of S on opposite sides of a given chord, their geodesic bisector (formed by concatenating their bisector and hyperbolic arcs) can cross the chord $\Theta(r)$ times. See Figs. 2 to 4 in Appendix C.

Section 2 describes the preprocessing procedures and data structures used by our algorithms. Section 3 discusses how we use a technique known as *parametric search* to solve the CCOSKEG disc problem. Section 4 summarizes our result. Appendices A and B contain details omitted from the paper due to space constraints. Appendix C contains figures illustrating some concepts from the paper.

2 Preprocessing, Data Structures, and Definitions

We perform the following preprocessing in $O(m)$ time and space.

Polygon Simplification Convert P_{in} into a simplified polygon P consisting of $O(r)$ vertices using the $O(m)$ time and space algorithm of Aichholzer et al. [5] that computes a polygon P such that: $P \supseteq P_{in}$; $|P|$ is $O(r)$; the reflex vertices in P_{in} also appear in P; P preserves the visibility of points in P_{in}; and the shortest path between two points in P_{in} remains unchanged in P. As with P_{in}, we assume the points of S are in general position with the vertices of P, and no four points of S are geodesically co-circular in P.

Shortest-Path Data Structure We use the $O(r)$ time and space algorithm of Guibas and Hershberger [29, 34] on P to build a data structure that gives the length of the shortest path between any two query points in P in $O(\log r)$ time and space. Querying the data structure with two points in P returns a tree of $O(\log r)$ height whose in-order traversal is the shortest path in P between the two query points. The query also provides the length of the path from the source to each node along the path (which is stored at the respective node in the tree). This data structure can provide the first or last edge along the path between the two points in $O(\log r)$ time by traversing the tree to a leaf. We can also perform a search through this tree to find the midpoint of the shortest path in $O(\log r)$ time [57, Lemma 3]. The returned tree has $O(r)$ nodes and edges, but the query adds $O(\log r)$ nodes and edges linking to pre-computed structures to produce the result.

Funnel [29, 37] The vertices of the geodesic shortest path $\Pi(a, b)$ are the vertices a, b, and a subset of the vertices of the polygon P forming a polygonal chain [20, 38]. Consider a diagonal ℓ of P, its two endpoints ℓ_1 and ℓ_2, and a point p in P. The union of the three paths $\Pi(p, \ell_1)$, $\Pi(p, \ell_2)$, and ℓ form what is called a funnel. This funnel represents the shortest paths from p to the points on ℓ in that their union is the funnel. Starting at p, the paths $\Pi(p, \ell_1)$ and $\Pi(p, \ell_2)$ may overlap during a subpath, but there is a unique vertex p_a called the *apex* (which is the farthest vertex on their common subpath from p) where the two paths diverge. After they diverge, the two paths never meet again. The path from p_a to an endpoint of ℓ forms an *inward-convex* polygonal chain (i.e., a convex path through vertices of P with the bend protruding into the interior of P). In our paper we often make use of the portion of the funnel between the apex and ℓ, which we shall refer to as a *truncated funnel.* Fig. 5 in Appendix C illustrates the notion of a funnel.

Definition 1 (Aronov 1989 [9, Definition 3.1]) *For any two points u and v of P, the last vertex (or u if there is none) before v on $\Pi(u, v)$ is referred to as the **anchor** of v (with respect to u).*

Guibas and Hershberger [29] and Oh and Ahn [45] point out that given the trees representing the shortest paths between a fixed source and two distinct destination points on the same chord, the apex of their funnel can be computed in $O(\log r)$ time.

Observation 1 *The apex of a funnel from a source point in P to the diagonal ℓ can be computed in $O(\log r)$ time and $O(r)$ space. The distance from the source point to the apex can also be determined in $O(\log r)$ time and $O(r)$ space.*

Distance Function of a Point $u \in S$: Let us review the graph we get by plotting the distance from a point u to a line ℓ where the position along ℓ is parameterized by x. Abusing notation, we call the x-monotone curve representing this graph the *distance function*, which we denote by $\text{dist}_u(\cdot)$. The domain of this function is ℓ and it returns the geodesic distance from u to $x \in \ell$ where x is the input of the function. Without loss of generality, we can assume that the x-axis is the line in question. For a point u, u_x is the x-coordinate of u and u_y is its y-coordinate. This distance function is actually a branch of a *right hyperbola*[6] whose eccentricity is $\sqrt{2}$ and whose focus is therefore at $\sqrt{2} \cdot u_y$. In our polygon P, $\text{dist}_u(\cdot)$ is a continuous piecewise hyperbolic function. If the funnel from u to the endpoints ℓ_1 and ℓ_2 of ℓ is trivial (i.e., a Euclidean triangle), then $\text{dist}_u(\cdot)$ has one piece expressed as $\text{dist}_u(x) = \sqrt{(x - u_x)^2 + u_y^2}$. If there are reflex vertices of P in u's funnel, $\text{dist}_u(\cdot)$ has multiple pieces. The formula for each piece is $\text{dist}_u(x) = \sqrt{(x - w_x)^2 + w_y^2} + d_g(u, w)$, where w is the anchor, and the **domain** of this hyperbolic piece is the set of values of x for which w is the anchor. Refer to Fig. 6 in Appendix C for an example of a multi-piece distance function.

[6]Also called a *rectangular* or *equilateral* hyperbola.

Definition 2 (Aronov 1989 [9, Definition 3.7]) *The* shortest-path tree *of P from a point s of P, $T(P, s)$, is the union of the geodesic shortest paths from s to vertices of P.*

Definition 3 (Aronov 1989 [9, between 3.8 and 3.9]) *Let e be an edge of $T(P, s)$ and let its endpoint furthest from s be v. Let $\overrightarrow{h}$ be the open half-line collinear with e and extending from v in the direction of increasing distance from s. If some initial section of $\overrightarrow{h}$ is contained in the interior of P, we will refer to the maximal such initial section as the* **extension segment** *of e.*

Definition 4 (Aronov 1989 [9, Definition 3.9]) *Let the collection of extension segments of edges of $T(P, s)$ be denoted by $E(P, s)$ (also simplified to E when the polygon and point are clear from the context).*

Consider the subset $E \subseteq E(P, u)$ whose elements define the domains of the pieces of $\text{dist}_u(\cdot)$ along ℓ. We refer to the intersection of an element of this set with ℓ as a *marker.* Sometimes we will need to identify domains that have specific properties so that an appropriate hyperbolic piece of $\text{dist}_u(\cdot)$ can be analyzed. Similar to other papers that find intervals of interest along shortest paths and chords [1, 2, 8, 45], we can use the funnel between u and ℓ to perform a binary search among the domain markers to find a domain of interest. Since domain markers are points along ℓ, one way they can be used is to provide distances away from u to compare against. We have the following observation.

Observation 2 *For an extension segment $e \in E$, if it takes $O(1)$ time and space to determine which side of $\ell \cap e$ contains a domain of interest along ℓ, then we can find a domain of interest along ℓ and its corresponding hyperbolic piece of $\text{dist}_u(\cdot)$ in $O(\log r)$ time and $O(r)$ space.*

3 CCOSKEG Disc: Parametric Search

Refer to Appendix A for missing details. Let $\partial D(u, \rho)$ denote the boundary of the geodesic disc centred at u with radius ρ. We use parametric search to find a SKEG disc centred on the chord ℓ.

Parametric search is a technique introduced by Megiddo [41, 42] for optimizing a numeric parameter through deduction using two algorithms in tandem. The first is a sequential *decision* algorithm. Given a candidate for the optimal value, the decision algorithm determines how this candidate relates to the optimal value (i.e., it determines whether the candidate is less than, equal to, or greater than the optimal value). Testing a candidate using the decision algorithm is usually costly, which is why the problem and the candidates need to have the following *monotonicity* property: if the test reveals that the optimum is greater (less) than the candidate tested, then it is also greater (less) than everything less (greater) than the tested candidate.

The second algorithm used is a parallel *generic* algorithm. This parallel algorithm (which is usually converted back into a sequential algorithm) typically solves a problem using comparisons whose outcomes depend on the parameter being optimized, or, in other words, comparisons of objects that *would* result if the optimal value were given. In a way, we work backwards by examining which properties/objects would exist if we had the optimum as well as how these objects would relate to each other. For example, our algorithm to solve the CCOSKEG disc problem sorts, along ℓ, $\partial D(u, \rho^*) \cap \ell$ for all $u \in S$. Using sorting algorithms as the generic algorithm has been done before [21, 28, 42, 52, 54]. See Figs. 7 and 8 in Appendix C.

The comparisons in the generic sorting algorithm are typically expressed as a polynomial equation featuring the parameter to be optimized as a variable in the equation. Refer to Fig. 9 in Appendix C. The comparison is resolved by computing the sign of the equation (i.e., positive, negative, or zero) given a value for the parameter. Each of these polynomial equations has roots that together form the sortable set of candidates for the optimal value. Parametric search uses the decision algorithm to test the candidates. As more of the relations of the candidates to the optimum are determined, more comparisons in the generic algorithm can be resolved. In this way we are able to eventually deduce the optimal value.

Either ρ^* will coincide with the closest distance of ℓ to some point in S; or at least two of the intersection points from distinct discs will coincide and hence ρ^* will be a root for some pair of equations. See Fig. 10 in Appendix C. When comparing two of these intersections/equations, to get our candidate radii through which we search for ρ^* we set the equations equal to each other and solve for the roots (which is where they have coinciding intersection points along ℓ). For the pair of intersection points that created a given set of roots, the roots create intervals in the parameter space (see Fig. 11 in Appendix C). Given the equation for a pair from which we extracted the roots, plugging in any value for the radius that lies in one of these root-defined intervals results in the equation having the same sign (either positive or negative), and thus the intersection points having the same order along ℓ.

The sorting algorithm proceeds until it cannot continue without resolving any comparisons (i.e., until it gets stuck). Being a parallel algorithm (or a sequentialized version of a parallel algorithm), the comparisons in one parallel step are all independent and present us with a set of candidate radii. The decision algorithm is run on a judiciously-chosen candidate radius followed by a cull of the remaining candidates that we infer are too large or small. This is repeated until some comparison can be resolved, at which point the algorithm proceeds until it again becomes stuck. Eventually, the relation of ρ^* to all of the roots in the candidate set is known.

The (at most) two intersection points of a disc with ℓ tell us where the intersection of a disc with ℓ begins and ends. We are interested in overlapping intervals of at least k discs.

3.1 Preliminaries

3.1.1 Testing Closest Points

We precompute, for each point $u \in S$, the closest point of ℓ to u, also known as the projection of u onto ℓ (see Fig. 12 in Appendix C for an example of projections). Let u_c be the closest point of ℓ to u. Equivalently, u_c is the point along ℓ that minimizes $\text{dist}_u(\cdot)$. We showed the following in our previous work [15].

Lemma 1 (Bose et al. 2023 **[15])** *We compute the set of projections of the points of S onto ℓ in $O(n \log r)$ time and $O(n + r)$ space.*

We are looking for ρ^*, the smallest radius of a KEG disc centred on ℓ, and the centre of such a disc. If a CCOSKEG disc has only one point $u \in S$ on its boundary, then a CCOSKEG disc is centred at the projection u_c and $\rho^* = d_g(u, u_c)$. Each of the n closest distances defined by projections is a candidate radius. To effectively perform a binary search among these candidates, we repeatedly perform an $O(n)$ time and space median selection algorithm on these radii [6, 13, 22]. In each iteration, we find the median, test it with the decision algorithm, then cull the remaining candidates now known to not be the optimum. Since we halve the number of elements to consider in each round, we perform $O(\log n)$ rounds and make $O(\log n)$ calls to the decision algorithm. The overall time spent over the $O(\log n)$ rounds performing median selections and culling the list is $O(n)$. After the search has finished, either we have found ρ^*, or we know that there will be at least two points of S on the boundary of a CCOSKEG disc.

Corollary 2 *With $O(\log n)$ calls to the decision algorithm and additional $O(n \log r)$ time and $O(n+r)$ space, we either compute ρ^* and a point c^* along ℓ such that $D(c^*, \rho^*)$ is a CCOSKEG disc of the points of S along*

ℓ, or we conclude that there are at least two points of S on the boundary of a CCOSKEG disc.

3.1.2 How to Compare Elements in the Sort

We will sort $\partial D(u, \rho^*) \cap \ell$ for all $u \in S$. We need to express these intersection points in terms of the variable radius ρ of the discs centred at the points of S. Assume that $\partial D(u, \rho)$ intersects ℓ twice (the cases of one and zero intersections are omitted). Assume that we know that the point $w = (w_x, w_y)$ is the last reflex vertex on the path from $u \in S$ to at least one of the intersection points. Let $\Delta = \rho - d_g(u, w)$. The equation for the circular arc defining $\partial D(u, \rho)$ where it intersects ℓ is given by the equation of a circle of radius Δ centred at w. Using the equation of a circle, we have the following.

$$x = \left(\pm\sqrt{\Delta^2 - w_y^2}\right) + w_x \tag{1}$$

If x is defined, it is only valid in the domain of w. If x is undefined, then after passing w, $D(u, \rho)$ does not intersect ℓ. We will assume we know the last reflex vertex before every intersection point and thus the $O(n)$ equations to use for the intersection points of the discs with ℓ. We show in Section 3.4 that we can use a parametric-search-like approach to find these $O(n)$ reflex vertices using an idea similar to one of the steps of the Goodrich and Pszona parametric search paper [28].

Now that we have our items to be sorted, we need to know how to compare them. Consider two of these intersection points, one for each of the points u and v, $\{u, v\} \in S$. Let the reflex vertex of the intersection point of u (resp. v) being considered be w (resp. z) and let the intersection points considered be the ones computed by taking the positive square roots in their equations (from Eq. (1)). Let $\delta = d_g(u, w)$ and $\psi = d_g(v, z)$. When we sort the intersection points, we are asking if one x-value is less than, greater than, or equal to another along ℓ. Therefore, we want to know the following at a variable radius ρ.

$$\sqrt{(\rho - \psi)^2 - z_y^2} + z_x \lesseqgtr \sqrt{(\rho - \delta)^2 - w_y^2} + w_x \tag{2}$$

We can expand and simplify Eq. (2) to get a cubic function replacing constant expressions by constant C_i.

$$0 \lesseqgtr C_1\rho^3 + C_2\rho^2 + C_3\rho + C_4 \tag{3}$$

The sign of the answer of Eq. (3) reveals which intersection point is to the left. Eq. (3) gives us a polynomial in ρ which determines the comparisons of the parallel sorting algorithm and whose roots are the candidates with which to run the decision algorithm. The roots are the values for which the two intersection points coincide. Once it is known to which side of each root the optimal ρ^* lies for this instance of Eq. (3), we know the result of the comparison for ρ^* for this instance.

3.2 Decision Problem

Lemma 3 *Given a polygon P with $O(r)$ vertices, a chord ℓ, a set S of n points, a radius ρ, and a constant $k \le n$, having performed the preprocessing of Lemma 1, we can decide if there is a KEG disc of radius ρ centred on ℓ and return such a disc in $O(n(\log r + \log n))$ time and $O(n + r)$ space, and report whether $\rho < \rho^*$, $\rho > \rho^*$, or $\rho = \rho^*$.*

Proof. [Sketch] In $O(\log r)$ time and $O(r)$ space we can build the two funnels of u between u_c and the endpoints of ℓ and then perform a binary search in each to locate the domain in which a point at distance ρ lies. This tells us which reflex vertex to use in Eq. (1). Thus, in $O(n \log r)$ time and $O(n + r)$ space, we create $O(n)$ labelled intervals: $\{D(u, \rho) \cap \ell : u \in S\}$. We then sort the interval endpoints in $O(n \log n)$ time and $O(n)$ space, and then walk along ℓ and count the maximum number of discs we are concurrently in at any given point. If the maximum is smaller than k, then ρ is too small. If the maximum is larger than k, then ρ is too large. If the maximum is k and there is a subinterval that is larger than a single point in which there are k overlapping intervals, then ρ is too large. Otherwise, $\rho = \rho^*$ and the single point of k overlaps is the centre for a CCOSKEG disc. □

3.3 Using Boxsort

Goodrich and Pszona [28] show that boxsort [50] can be used as our sorting algorithm. It can be described as quicksort with multiple pivots which produces a number of recursive calls proportional to the number of pivots. See Fig. 13 in Appendix C for an illustrated example of a recursive call. This allows them to take advantage of the optimization technique of Cole [21] to reduce the running time.

Theorem 4 *Given a chord $\ell \subset P_{in}$ we compute a CCOSKEG disc $D(c^*, \rho^*)$ in $O(n \log^2 n + m)$ time with high probability using $O(n \log r + m)$ space.*

Proof. [Sketch] Preprocessing from Section 2 takes $O(m)$ time and space. It will be shown in Section 3.4 that with $O(\log n + \log r)$ calls to the decision algorithm and additional $O(n \log r)$ time and $O(n \log r + r)$ space, we compute the last reflex vertices on the paths from each point $u \in S$ to $\partial D(u, \rho^*) \cap \ell$, effectively giving us $O(n)$ items to sort. Given this result and Corollary 2, the preprocessing from Section 3.1 makes $O(\log n + \log r)$ calls to the decision algorithm of Lemma 3, uses $O(n \log r + r)$ space, and takes time $O(n \log n \log r + n \log^2 n + n \log^2 r)$.

As seen in Goodrich and Pszona [28], Motwani and Raghavan [44], and Reischuk [50], with high probability (i.e., at least $1 - e^{-\log^b n}$ for some constant $b > 0$) boxsort chooses a "good" sequence of pivots so that it only requires $O(\log n)$ calls to the decision algorithm of Lemma 3; and with the same probability, taking into account the number of recursive calls and the time we spend in a recursive call to create boxes and then sort the remaining comparisons into their boxes, using boxsort for parametric search takes $O(n \log n + \log n \cdot n(\log r + \log n)))$ time and $O(n + r)$ space.

Considering the $O(m)$ time spent in preprocessing, we can simplify the runtime to $O(n \log^2 n + m)$ with high probability by assuming some terms are dominant and arriving at a contradiction. The overall space used is $O(n \log r + m)$. □

3.4 Decreasing to a Linear Number of Items to Sort

In this section, our goal is to discover which $O(n)$ reflex vertices to use for Eq. (1) for the points of S. The procedure (and its analysis) is like one of the steps used in the boxsort parametric search of Goodrich and Pszona [28]. Similar to routing unsorted elements through the binary tree of sorted pivots to find their "box" for the next recursive call, we independently route through $2n$ binary search trees of $O(\log r)$ height, where the outcomes of the comparisons depend on the solution to the parametric search. This allows us to find the reflex vertices for each $u \in S$ that anchor $\partial D(u, \rho^*) \cap \ell$. However, instead of inferring the optimum by sorting intersection points described as equations from which candidates for the optimum are extracted, here our comparisons are directly in the parameter space: we are directly comparing distances against the optimum. We illustrate an example in Fig. 14 in Appendix C.

Since domain markers for the funnel of a point in S with ℓ are points along ℓ, in addition to defining domains for reflex vertices they also provide distances to use as candidate radii. We have the following monotonicity property: for any point $u \in S$, the distance to ℓ increases monotonically as we move from its closest point $u_c \in \ell$ to the endpoints of ℓ [48]. Thus, if we can decide how the radius produced by a given marker compares to the optimal radius, we can perform two binary searches (recall Observation 2) among these markers between u_c and the endpoints of ℓ to find the domains which contain $\partial D(u, \rho^*) \cap \ell$, and hence discover which reflex vertices to use in Eq. (1) for u in the main parametric search.

We sequentialize the running of $2n$ parallel searches through binary trees of $O(\log r)$ height (one per funnel). Routing an element through these search trees is similar to following a directed path of $O(\log r)$ height. For each point $u \in S$ we search through the domain markers implicitly contained in its two funnels looking for the domains that contain $\partial D(u, \rho^*) \cap \ell$, which are the domains that contain a point that is distance ρ^* away from u.

For each tree through which we are routing, each step produces a comparison to resolve. Since a call to the decision algorithm is considered costly, we do not want to call the decision algorithm to resolve each comparison individually. Using the fact that the candidate radii have the monotonicity property we need for parametric search (i.e., given the relation between ρ^* and a candidate radius, we know the relation between ρ^* and either everything bigger than or less than the candidate) and the fact that the domain markers used in the comparisons of the searches also provide candidate radii, we can route through the trees with a logarithmic number of calls to the decision algorithm. Following Goodrich and Pszona [28], we assign weights to the comparisons in the searches. The routing can be considered as iterations involving three steps: in the first step, we use a linear-time weighted-median-finding algorithm [51] to choose the weighted median candidate radius; in the next step, we input that radius into the decision algorithm; when the decision algorithm returns, the last step is to repeatedly resolve all active comparisons that can be resolved until no more routing can be performed without knowing the result of another call to the decision algorithm. At this point, the next iteration begins.

Lemma 5 (Cole 1987 **[21], Goodrich and Pszona** 2013 **[28])** *For $j \geq 5(i + (1/2)\log(4n))$, during the $(j+1)^{st}$ iteration there are no active comparisons at depth i.*

Plugging our $2n$ routing trees of height $O(\log r)$ into the analyses of Goodrich and Pszona [28] and Cole [21] yields that there are $O(\log n + \log r)$ calls to the decision algorithm.

Corollary 6 *With $O(\log n + \log r)$ calls to the decision algorithm, with additional $O(n \log r)$ time and $O(r + n \log r)$ space, we compute for each $u \in S$ the anchors of $\partial D(u, \rho^*) \cap \ell$.*

4 Conclusion

By using the result of Goodrich and Pszona [28], we were able to use boxsort [50] to implement parametric search to solve the CCOSKEG disc problem. Though a 2-approximation to a SKEG disc contains $\Theta(\min(n, kr))$ points of S in general, we can use Theorem 4 together with an exact smallest k-enclosing algorithm for planar instances [32] to find a radius ρ at most twice the optimal radius of a SKEG disc such that any disc with radius ρ contains at most $4k$ points of S (see Appendix B).

References

[1] Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. In *Symposium on Computational Geometry*, volume 99 of *LIPIcs*, pages 4:1–4:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[2] Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. *CoRR*, abs/1803.05765, 2018.

[3] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12(1):38–56, 1991.

[4] Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 56(4):836–859, 2016.

[5] Oswin Aichholzer, Thomas Hackl, Matias Korman, Alexander Pilz, and Birgit Vogtenhuber. Geodesic-preserving polygon simplification. *International Journal of Computational Geometry & Applications*, 24(4):307–324, 2014.

[6] Andrei Alexandrescu. Fast deterministic selection. In *SEA*, volume 75 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[7] Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. Linear-time triangulation of a simple polygon made easier via randomization. In *Symposium on Computational Geometry*, pages 201–212. ACM, 2000.

[8] Lars Arge and Frank Staals. Dynamic geodesic nearest neighbor searching in a simple polygon. *CoRR*, abs/1707.02961, 2017.

[9] Boris Aronov. On the geodesic voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1):109–140, 1989.

[10] Boris Aronov, Steven Fortune, and Gordon T. Wilfong. The furthest-site geodesic voronoi diagram. *Discrete & Computational Geometry*, 9:217–255, 1993.

[11] Tetsuo Asano and Godfried Toussaint. Computing the geodesic center of a simple polygon. In *Discrete Algorithms and Complexity*, pages 65–79. Elsevier, 1987.

[12] Sang Won Bae, Matias Korman, and Yoshio Okamoto. Computing the geodesic centers of a polygonal domain. *Comput. Geom.*, 77:3–9, 2019.

[13] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[14] Magdalene G. Borgelt, Marc J. van Kreveld, and Jun Luo. Geodesic disks and clustering in a simple polygon. *Int. J. Comput. Geometry Appl.*, 21(6):595–608, 2011.

[15] Prosenjit Bose, Anthony D'Angelo, and Stephane Durocher. Approximating the smallest k-enclosing geodesic disc in a simple polygon. In *WADS*, page (to appear). LNCS, 2023.

[16] Prosenjit Bose and Godfried T. Toussaint. Computing the constrained euclidean geodesic and link center of a simple polygon with application. In *Computer Graphics International*, pages 102–110. IEEE Computer Society, 1996.

[17] Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discret. Comput. Geom.*, 22(4):547–567, 1999.

[18] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.

[19] Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[20] Orin Chein and Leon Steinberg. Routing past unions of disjoint linear barriers. *Networks*, 13(3):389–398, 1983.

[21] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

[23] Anthony D'Angelo. *Constrained Geometric Optimization Problems.* PhD thesis, Carleton University, 2023. `doi:10.22215/etd/2023-15445`.

[24] Amitava Datta, Hans-Peter Lenhof, Christian Schwarz, and Michiel H. M. Smid. Static and dynamic algorithms for k-point clustering problems. *J. Algorithms*, 19(3):474–503, 1995.

[25] Sarita de Berg and Frank Staals. Dynamic data structures for k-nearest neighbor queries. *Computational Geometry*, 111:101976, 2023.

[26] Alon Efrat, Micha Sharir, and Alon Ziv. Computing the smallest k-enclosing circle and related problems. *Comput. Geom.*, 4:119–136, 1994.

[27] David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11:321–350, 1994.

[28] Michael T. Goodrich and Pawel Pszona. Cole's parametric search technique made practical. In *CCCG*. Carleton University, Ottawa, Canada, 2013.

[29] Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

[30] Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[31] Sariel Har-Peled. *Geometric approximation algorithms*, volume 173. American Mathematical Soc., 2011.

[32] Sariel Har-Peled and Soham Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

[33] Juha Heinonen. *Lectures on analysis on metric spaces*. Springer, New York, 2001.

[34] John Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991.

[35] John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995.

[36] David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[37] Der-Tsai Lee and Franco P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[38] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

[39] Jiří Matoušek. On enclosing k points by a circle. *Inf. Process. Lett.*, 53(4):217–221, 1995.

[40] Jiří Matoušek. On geometric optimization with few violated constraints. *Discrete & Computational Geometry*, 14(4):365–384, 1995.

[41] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.

[42] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[43] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983.

[44] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[45] Eunjin Oh and Hee-Kap Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete & Computational Geometry*, 63(2):418–454, 2020.

[46] Eunjin Oh, Sang Won Bae, and Hee-Kap Ahn. Computing a geodesic two-center of points in a simple polygon. *Comput. Geom.*, 82:45–59, 2019.

[47] Eunjin Oh, Jean-Lou De Carufel, and Hee-Kap Ahn. The geodesic 2-center problem in a simple polygon. *Comput. Geom.*, 74:21–37, 2018.

[48] Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4:611–626, 1989.

[49] George Rabanca and Ivo Vigan. Covering the boundary of a simple polygon with geodesic unit disks. *CoRR*, abs/1407.0614, 2014.

[50] Rüdiger Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM J. Comput.*, 14(2):396–409, 1985.

[51] Angelika Reiser. A linear selection algorithm for sets of elements with weights. *Inf. Process. Lett.*, 7(3):159–162, 1978.

[52] Sivan Toledo. *Extremal polygon containment problems and other issues in parametric searching*. PhD thesis, Citeseer, 1991.

[53] G Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle*, 3(2):9–42, 1989.

[54] René van Oostrum and Remco C. Veltkamp. Parametric search made practical. *Comput. Geom.*, 28(2-3):75–88, 2004.

[55] Ivo Vigan. Packing and covering a polygon with geodesic disks. *CoRR*, abs/1311.6033, 2013.

[56] Haitao Wang. On the geodesic centers of polygonal domains. *JoCG*, 9(1):131–190, 2018.

[57] Haitao Wang. An optimal deterministic algorithm for geodesic farthest-point voronoi diagrams in simple polygons. In *SoCG*, volume 189 of *LIPIcs*, pages 59:1–59:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[58] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 1991.

A Parametric Search

Let $\partial D(u, \rho)$ denote the boundary of the geodesic disc centred at u with radius ρ. We assume preprocessing has already been performed. Note that the initial input chord ℓ of P_{in} may no longer be a chord in our simplified polygon P. We continue to use the initial chord since (a) shortest paths between points in P_{in} don't change when P_{in} is simplified to P; and (b) the endpoints of our given chord would define an interval of solution validity anyway if we chose to extend it into a chord for P (which could be done in $O(\log r)$ time and $O(1)$ space using ray-shooting queries).

Ray-Shooting Queries In $O(r)$ time and space we preprocess P to allow us to perform $O(\log r)$-time, $O(1)$-space ray-shooting queries that take as input a point in P and a direction and returns the point on ∂P (i.e., the boundary of P) where the ray first intersects ∂P [19, 35].

Remark 1 *It is not clear whether it is possible to apply the simpler recursive random sampling technique of Chan's that rivals parametric search to solve the CCOSKEG disc problem [17]. That approach requires one to partition the points of S into a constant number of fractional-sized subsets such that the overall solution is the best of the solutions of each of the subsets. It is not clear to us how to partition the points of S to take advantage of this approach.*

A.1 Testing Closest Points

Lemma 7 (Bose et al. 2023 [**15**]**)** *We compute the set of projections of the points of S onto ℓ in $O(n \log r)$ time and $O(n + r)$ space.*

Proof. Let ℓ be horizontal. For ease of presentation, we consider ℓ as having subdivided P into two polygons. We consider one of these polygons, let it keep the name P, and assume the points of S are in P. The other subpolygon can then be analyzed identically. Let the downward direction be toward the side of ℓ containing the exterior of the polygon P. Let the left endpoint of ℓ be ℓ_1 and the right endpoint be ℓ_2. Consider a point $p \in \ell$ and the last edge e of $\Pi(u, p)$ (i.e., the edge to which p is incident). Let the *angle of* e be the smaller of the two angles formed by e and ℓ at p. The range of this angle is $[0, \pi/2]$. We know from Pollack et al. [48, Corollary 2] that $\mathrm{dist}_u(\cdot)$ is minimized when e is perpendicular to ℓ. We also know from Pollack et al. [48, Corollary 2] that given $p' \in \ell$ and an edge e' analogous to e, if the angle of e' is closer to $\pi/2$ than that of e, then $\mathrm{dist}_u(p') < \mathrm{dist}_u(p)$. Lastly, we know from Pollack et al. [48, Lemma 1] that $\mathrm{dist}_u(\cdot)$ is a convex function which means it has a global minimum.

Using Observations 1 and 2 we can retrieve the truncated funnel of u and ℓ in $O(\log r)$ time and $O(r)$ space and use the convex chains to perform a binary search along ℓ using the markers defined by the elements of E to find the domain in which u_c lies. See Fig. 5 in Appendix C. This domain has the property that the angle of the last edge on the shortest path from u to the points in this domain is closest to $\pi/2$.

In the binary search, at each marker (as determined by the node currently being visited in the tree representing the convex chain), in $O(1)$ time and space we compute the angle of ℓ with the extension segment defining the marker. Since $\mathrm{dist}_u(\cdot)$ is a convex function, we know that as we slide a point $p \in \ell$ from ℓ_1 to ℓ_2, the angle of the edge incident to p on $\Pi(u, p)$ will monotonically increase until it reaches $\pi/2$, then monotonically decrease. Thus, after computing the angle of the extension segment with ℓ, we know to which side of its marker to continue our search: the side that contains the smaller angle (because moving in this direction will increase the smaller angle). Thus by Observation 2 the search takes $O(\log r)$ time and $O(r)$ space.

At the end of our search we will have the reflex vertex whose domain contains the edge that achieves the angle closest to $\pi/2$. Then in $O(1)$ time and space we can build the corresponding piece of $\mathrm{dist}_u(\cdot)$ and find the value along ℓ that minimizes it.

The space bounds follow from the n projections that are computed and the $O(r)$ space used by the shortest-path data structure queries. □

A.2 How to Compare Elements in the Sort

The trick when using a sorting algorithm as the generic algorithm in the parametric search technique is deciding what to sort. Once that has been determined, we use parametric search to run the sorting algorithm as if the things we are sorting were produced knowing ρ^*.

We will sort $\partial D(u, \rho^*) \cap \ell$ for all $u \in S$ (i.e., the intersection points of ℓ with the boundaries of the geodesic discs of radius ρ^* centred at the points of S). We need to express these intersection points in terms of the variable radius ρ of the discs centred at the points of S. Notice that the boundary of a geodesic disc of radius ρ is constructed piecewise. Part of the disc's boundary is formed by the boundary of the polygon at distance less than ρ away from the centre of the disc, and the rest is circular arcs from the circle centred at the disc's centre or from circles centred on reflex vertices contained in the disc's interior.

Let us assume for the moment that $\partial D(u, \rho)$ intersects ℓ twice (the cases of one and zero intersections are simple to figure out afterwards and are omitted). Assume that we know that the point $w = (w_x, w_y)$ is the last reflex vertex on the path from $u \in S$ to at least one of the intersection points. This intersection point is where ℓ is intersected by a circular arc centred on w. Let $\Delta = \rho - d_g(u, w)$. If Δ were negative, we would have a contradiction ($D(u, \rho)$ would not even contain w). The equation for the circular arc defining $\partial D(u, \rho)$ where it intersects ℓ is given by the equation of a circle of radius Δ centred at w. Once again, assume ℓ is the x-axis. Using the equation of a circle, we have the following.

$$\begin{aligned}
(x - w_x)^2 + (y - w_y)^2 &= \Delta^2 \\
(x - w_x)^2 + (0 - w_y)^2 &= \Delta^2 \\
(x - w_x)^2 + w_y^2 &= \Delta^2 \\
(x - w_x)^2 &= \Delta^2 - w_y^2 \\
x - w_x &= \pm\sqrt{\Delta^2 - w_y^2} \\
x &= \left(\pm\sqrt{\Delta^2 - w_y^2}\right) + w_x \qquad (4)
\end{aligned}$$

If x is defined, it is only valid in the domain of w (i.e., the interval along ℓ in which w is the last reflex vertex on the path from u). If x is undefined, then after passing w, $D(u, \rho)$ does not intersect ℓ. If both values computed by Eq. (4) fall outside of w's domain, then we contradict that w is the last vertex on the path from u to the considered intersection point for the given radius ρ (which means that w would not be used in computing the boundary of $D(u, \rho)$). Otherwise, if an x-value from Eq. (4) lies within the domain of w, then this x-value would be one of at most two intersection points of $\partial D(u, \rho)$ and ℓ. If only one x-value computed by Eq. (4) falls in the domain of w, then the process must be repeated with some other reflex vertex (which is the case if the last reflex vertex from u is not the same for both intersection points).

Though we want to sort intersection points of ℓ with the boundaries of geodesic discs, our intersection points are equations until the variable ρ has been provided. Nonetheless, it is these intersection points we would like to sort. Ideally, we would have only $O(n)$ candidate intersection points along ℓ to consider (up to two per $u \in S$). As we saw above though, the intersection points of a geodesic disc centred on $u \in S$ depend on the last reflex vertex on the path from u to ℓ, which in turn depends on the optimal radius, which we do not know ahead of time. Initially, it seems that for each $u \in S$ we have to consider the $O(r)$ intersection points computed by using each reflex vertex of its truncated funnel. However, we do not want to spend $\Omega(nr)$ time. Luckily for us, as we show in Appendix A.5, we can use a parametric-search-like approach to whittle these $O(nr)$ candidates back down to $O(n)$ using an idea similar to one of the steps of the Goodrich and Pszona parametric search paper [28]. We will assume we know the last reflex vertex before every intersection point and thus the $O(n)$ equations to use for the intersection points of the discs with ℓ.

Now that we have our items to be sorted, we need to know how to compare them. Consider two of these intersection points, one for each of the points u and v, $\{u, v\} \in S$. Let the reflex vertex of the intersection point of u (resp. v) being considered be w (resp. z) and let the intersection points considered be the ones computed by taking the positive square roots in their equations (from Eq. (4)). Let $\delta = d_g(u, w)$ and $\psi = d_g(v, z)$. When we sort the intersection points, we are asking if one x-value is less than, greater than, or equal to another along ℓ. Therefore, we want to know the following at a variable radius ρ. Let C_i be constant i.

$$\begin{aligned}
& \sqrt{(\rho - \psi)^2 - z_y^2} + z_x \lesseqqgtr \sqrt{(\rho - \delta)^2 - w_y^2} + w_x \\
\Rightarrow \quad & 0 \lesseqqgtr (\rho - \delta)^4 + (\rho - \psi)^4 \\
& \quad - 2(\rho - \delta)^2(\rho - \psi)^2 \\
& \quad + C_1(\rho - \delta)^2 + C_2(\rho - \psi)^2 \\
& \quad + C_3 \qquad (5)
\end{aligned}$$

We can expand and simplify Eq. (5) to get a cubic function, once again replacing constant expressions by constant C_i.

$$0 \lesseqqgtr C_4\rho^3 + C_5\rho^2 + C_6\rho + C_7 \qquad (6)$$

We end up with the cubic Eq. (6). After testing the projections of S onto ℓ in Section 3.1.1/Appendix A.1, we know and discard the points of S too far from ℓ to intersect ℓ with a disc of radius ρ^*. Thus, Eq. (4) will be defined at radius ρ^* for each point being considered, and the abscissa will be in the domain of the associated reflex vertex. Thus, when the comparison of Eq. (6) is resolved, a value for the radius is used that: (a) produces the same sign as ρ^*; and (b) adheres to the restriction that the results of using that radius with the two instances of Eq. (4) that created the comparison lie in the respective domains (along ℓ) of the reflex vertices associated with the instances of Eq. (4). The sign of the answer reveals which intersection point is to the left. When the comparison is resolved in the parametric search, both intersection points are defined and valid.

Eq. (6) gives us the next piece of the parametric search puzzle: a low-degree polynomial in ρ which determines the comparisons of the parallel sorting algorithm and whose roots are the candidates with which to run the decision algorithm. The roots are the values for which the two intersection points coincide. As mentioned above, if ρ^* is not defined by the closest point of ℓ to some point in S, then at ρ^* there will be at least one pair of intersection points that coincide since the overlapping interval of the $\geq k$ discs along ℓ will collapse to a single point. The constant number of roots for an instance of Eq. (6), which can be computed in $O(1)$ time and space since it is a cubic function, split the possible values of ρ for that instance into a constant number of intervals in the parameter space. Each interval has the property that evaluating the instance of Eq. (6) using any value of ρ in the interval produces the same sign. Therefore, once it is known to which side of each root the optimal ρ^* lies for this instance of Eq. (6), we know the result of the comparison for ρ^* for this instance.

A.3 Decision Problem

To use parametric search, we need a sequential decision algorithm that, given a radius as a candidate for ρ^*, can tell us if this candidate is less than, greater than, or equal to ρ^*.

Lemma 8 *Given a polygon P with $O(r)$ vertices, a chord ℓ, a set S of n points, a radius ρ, and a constant $k \leq n$, having performed the preprocessing of Lemma 1, we can decide if there is a KEG disc of radius ρ centred on ℓ and return such a disc in $O(n(\log r + \log n))$ time and $O(n + r)$ space, and report whether $\rho < \rho^*$, $\rho > \rho^*$, or $\rho = \rho^*$.*

Proof. Consider the geodesic disc of radius ρ, $D(u, \rho)$. Since the disc is geodesically convex, if the chord intersects the disc in only one point, it will be at the projection u_c. If it does not intersect the disc, then at u_c the distance from u to ℓ will be larger than ρ. Otherwise, if the chord intersects the disc in two points, u_c splits ℓ up into two intervals, each with one intersection point (i.e., each one contains a point of $\partial D(u, \rho) \cap \ell$). If u_c is an endpoint of ℓ, assuming ℓ has positive length, one of these intervals may degenerate into a point, making u_c coincide with one of the intersection points.

These two intervals to either side of u_c have the property that on one side of the intersection point contained within, the distance from u to ℓ is larger than ρ, and on the other side, the distance is less than ρ. Therefore, if $\partial D(u, \rho)$ does intersect ℓ in two points we can proceed as in the proof of Lemma 1: in $O(\log r)$ time and $O(r)$ space we can build the two funnels of u between u_c and the endpoints of ℓ (truncated at the apices) and then perform a binary search in each to locate the domain in which a point at distance ρ lies. We find the subinterval delimited by the domain markers of the reflex vertices wherein the distance from u to ℓ changes from being more (less) than ρ to being less (more) than ρ. The final subinterval for a given intersection point tells us which reflex vertex to use in Eq. (1). Once we find this domain, we can compute $\partial D(u, \rho) \cap \ell$ in $O(1)$ time and space. Thus, in $O(n \log r)$ time and $O(n + r)$ space, we create $O(n)$ labelled intervals along ℓ, one for each geodesic disc of radius ρ centred on each $u \in S$. In other words, the set of these intervals is $\{D(u, \rho) \cap \ell : u \in S\}$. We then sort these interval endpoints in $O(n \log n)$ time and $O(n)$ space, associating each endpoint with the interval it opens or closes.

When we walk along ℓ and enter the interval $D(u, \rho) \cap \ell$ for some $u \in S$, we say we are in the disc of u. Our next step, done in $O(n)$ time and space, is to walk along ℓ and count the maximum number of discs we are concurrently in at any given point. In other words, we are counting the maximum number of overlapping intervals. If the maximum is fewer than k, then ρ is too small. If the maximum is larger than k, then since we assume no four points are co-circular (and thus the CCOSKEG disc contains exactly k points), ρ is too large. If the maximum is k, if there is a subinterval that is larger than a single point in which there are k overlapping intervals, then ρ is too large. Otherwise, $\rho = \rho^*$ and the single point of k overlaps is the centre for the CCOSKEG disc. $\square$

A.4 Using Boxsort

Goodrich and Pszona [28] use boxsort [50] as their sorting algorithm. It can be described as quicksort with multiple pivots which produces a number of recursive calls proportional to the number of pivots. This allows them to take advantage of the optimization technique of Cole [21] to reduce the running time. Although the pivots first need to be sorted and then the remaining elements need to be sorted into the correct boxes (i.e., placed between the correct pivots) before recurring, we have multiple boxes once the recursive calls start. Each box has an independent set of comparisons (i.e., the comparisons in a box are independent of other boxes). This allows the recursion in the different boxes to be at different levels. Rather than running a median-finding algorithm on the *value* of the candidate radii, however, a weighting scheme for the candidate radii is applied based on the depth of their defining comparisons in the recursion. The next radius to test with the decision algorithm is based on a linear-time weighted-median-finding algorithm [51]. The sum of the weights of the current candidates is called the *active* weight. The weighted-median-finding algorithm considers the couples of radius and weight and returns the set of elements whose sum of weights is at most half the active weight. Furthermore, all radii in this set are less than the radius in the computed weighted median, and adding the weight of the computed weighted median produces a weight larger than half the active weight. The algorithm can easily be modified to return the weighted median as well. As such, rather than each call removing half of the *number* of candidate radii and comparisons, each call removes at least a quarter[7] of the

[7] Although the weighted median resolves the comparisons of a weighted *half* of the candidates, the weighting scheme applied by Goodrich and Pszona [28] equally assigns half of the weight of a comparison to its children. Thus, half of the active weight is

active weight.

The candidate radii are not separated by recursive subproblem; the weighted-median that gets resolved is chosen from the complete set of untested radii that have not already been culled. Within a recursive subproblem of the Goodrich and Pszona approach [28], however, there are "synchronization points" in the algorithm represented as "virtual comparisons" that are not activated until the current batch of comparisons has been resolved. These virtual comparisons do not represent real work, but they assist in the analysis of the runtime. The analysis is done by creating a dependency graph between the comparisons in the algorithm where the height of the graph of one recursive subproblem (i.e., the longest path between the start of the recursive call and the point when the next recursive calls start) is $O(\log n)$, and then noting that this implies the height for the entire simulation is also $O(\log n)$ with high probability.

Recall that the items we are sorting are the $O(n)$ intersection points of the boundaries of the candidate geodesic discs with ℓ. Call these points *crossings*. We repeat the algorithm of Goodrich and Pszona [28] that uses the following weighting rule for the comparisons (virtual or not). The following algorithm description (which does not mention the virtual comparisons as they do not represent real work) assumes each comparison produces one root. See Fig. 13 in Appendix C for an illustrated example of a recursive call.

Weight Rule When comparison $\mathbb{C}$ of weight σ gets resolved and causes q comparisons $\mathbb{C}_1, \ldots, \mathbb{C}_q$ to become active, each of these comparisons gets weight $\sigma/(2q)$.

1. Randomly mark $\sqrt{n}$ crossings.
2. Sort the marked crossings by comparing every pair in $O(n)$ comparisons, each of weight σ. Order them with insertion sort.
3. After all of the comparisons of the previous step have been resolved, *activate* comparisons for routing the remaining crossings through the *tree of marked items* (i.e., we do a binary search through the marked items), where each comparison at the root of this tree has weight $\sigma/(2n^2)$. In other words, create comparisons and assign the appropriate weight to them to prepare the $n - \sqrt{n}$ unmarked crossings for a binary search through the marked crossings to place the unmarked crossings between the marked pivots.
4. Route the unmarked crossings through the tree (i.e., do a binary search for each of them with the marked items) by repeatedly finding and testing the weighted median and then resolving comparisons (following the weighting rule when comparisons get resolved).
5. Once we know in which box each unmarked element lies (i.e., between which marked items it lies), insert it into its appropriate box.
6. Assign weight $\sigma/(4n^{4.5})$ to the initial comparisons in the new subproblems.
7. Recur into subproblems simultaneously.

removed, but if each comparison involved had children, then we add back a quarter of the weight (i.e., half of half).

The Goodrich and Pszona analysis [28] omits a discussion about comparisons with multiple roots and how the weights change in such cases. Below we alter their analysis to use three roots.

After the marked crossings are sorted, we use the sorted crossings to perform a binary search to position each unsorted element between a pair of sorted marked crossings. In the Goodrich and Pszona analysis [28], this is presented as a binary search through a perfectly balanced binary search tree for each unmarked element independently. To keep the analysis simple, rather than routing $n - \sqrt{n}$ items, we route n items. Our comparisons have three roots, so the weight of the comparison at the root of this binary search tree (which is the same for each element being routed) must change accordingly.

We begin the analysis. To sort the marked crossings by brute force, each comparison between a pair of crossings actually produces three comparisons of roots against the optimal radius. Each of these three comparisons started with weight σ. Thus, after these crossings are sorted, following the Goodrich and Pszona analysis [28] using an upside-down virtual binary tree of $\log(3n)$ height in the dependence graph, the weight at the root of the virtual tree is $\sigma/(3n)$. This virtual root then activates (and equally shares half of its weight to) the comparison nodes that start routing the unmarked crossings through the binary search tree of marked crossings. Each comparison at the root of these binary search trees that route the unmarked crossings through the search tree of marked crossings, however, also creates three root comparisons. Thus, each of these root comparisons gets weight $\sigma/(2 \cdot (3n)(3n)) = \sigma/(2(3n)^2)$ (i.e., weight $\sigma/(3n)$ divided among $3n$ comparisons).

As the routing progresses through the binary search trees, the trees get whittled down to paths determining where an element lies in relation to the sorted crossings. When each comparison in the tree produces one root comparison, the weight at the bottom of the tree is the weight at the top divided by $2^{\log \sqrt{n}} = n^{0.5}$ because each comparison along the way passes half its weight to its one child, i.e., the next comparison on the path through the tree. However in our scenario, although resolving a routing comparison in the tree activates at most one new routing comparison, it has three children, one for each root comparison of the next tree node. To aid in the analysis, we replace each routing comparison with the three root comparisons, all of which are the parents of a virtual comparison representing the routing comparison they resolve. Each of the three root comparisons of a routing node depend on (i.e., are children of) the virtual comparison of the node above it. In this way, rather than dividing the weight by half each step down the routing tree, we divide it by $2 \cdot (2 \cdot 3)$: each of the three root comparisons passes half of its weight to their (virtual) child (meaning it gets half the weight of any one of them), and this virtual node passes half of its weight equally shared amongst its three children, meaning each child gets half of a third of its weight. Thus, the weight at the bottom of our tree is the weight at the top divided by $(2 \cdot 2 \cdot 3)^{\log \sqrt{n}} = n \cdot 3^{\log \sqrt{n}}$. Although after the last routing comparison we do not create three new root comparisons, we create three virtual comparisons to make the analysis cleaner. Therefore, the weight at the bottom of the tree is $\sigma/(18n^2 \cdot n \cdot 3^{\log \sqrt{n}}) = \sigma/(18n^3 \cdot 3^{\log \sqrt{n}})$.

The next part of the dependence graph is another upside-down virtual binary tree like the one used after the sorting of the marked crossings. At the root of this tree, the weight becomes $\sigma/(18n^3 \cdot 3^{\log \sqrt{n}} \cdot 3n) = \sigma/(18n^4 \cdot 3^{\log \sqrt{n}+1})$. All initial comparisons in the subsequent recursive calls depend on the root of this tree and its weight. Thus the weight of the initial root comparisons in subsequent recursive calls is $\sigma/(2 \cdot (3n) \cdot (18n^4 \cdot 3^{\log \sqrt{n}+1})) = \sigma/(36n^5 \cdot 3^{\log \sqrt{n}+2})$ (i.e., weight $\sigma/(18n^4 \cdot 3^{\log \sqrt{n}+1})$ divided among $3n$ comparisons).

We can follow the approach of Goodrich and Pszona [28] and use Cole's analysis [21], which we repeat here modified for this specific case of at most three roots per comparison, to show that there are $O(\log n)$ calls to the decision algorithm.

Lemma 9 (Cole 1987 [21]) *At the start of the $(j+1)^{st}$ iteration, the active weight is bounded above by $(3/4)^j \cdot (3n)$ for $j \geq 0$.*

Proof. We prove the result by induction on j. At the start of the first iteration there are $3n$ active comparisons at depth 0, and all other comparisons are inactive. So for $j = 0$, the result holds. To prove the inductive step, it is sufficient to show that in each iteration the active weight is reduced by at least one quarter. We now show this.

Consider an active comparison $\mathbb{C}$ of weight σ that has just been resolved. Then $\mathbb{C}$ ceases to be active, and up to three new comparisons may become active, each an equal share of half the weight of σ (e.g., if three new comparisons are activated, they each have weight $\sigma/(2 \cdot 3) = \sigma/6$). So the resolution of $\mathbb{C}$ reduces the active weight by at least $\sigma/2$. Let the active weight be $\mathbb{W}$. In one iteration, we are guaranteed that the comparisons resolved have combined weight at least $\mathbb{W}/2$. Thus, in one iteration, the active weight is reduced from $\mathbb{W}$ to at most $3\mathbb{W}/4$. □

Lemma 10 (Goodrich and Pszona 2013 [28]) *Each comparison at depth i has weight $\geq (1/4)^i$.*

Proof. We prove this by induction on the depth of the boxsort recursion. Assume that the current recursive call operates on a subproblem of size $3n$, and that comparisons at the beginning of the recursive call have depth i and weight σ. By the inductive assumption, $\sigma \geq (1/4)^i$.

Consider comparisons in the current recursive call. Comparisons at depth j in the first tree of *virtual* comparisons (global depth $i + j$) have weight $\sigma/2^j \geq (1/4)^i \cdot (1/2)^j \geq (1/4)^{i+j}$. The last of them has (local) depth $\log(3n)$ and weight $\sigma/(3n)$. It then spreads half of its weight to $3n$ comparisons at depth $\log(3n) + 1$ (global depth $i + \log(3n) + 1$), setting their weight to

$$\sigma/(2(3n)^2) \geq \sigma/(4(3n)^2) = \sigma/(4^{\log(3n)+1}) \geq (1/4)^{i+\log(3n)+1}$$

The same reasoning follows for the case of the second *virtual* tree and recursive split.

Routing through the tree of sorted *marked* items has two levels of comparison nodes per node in the tree. For each step down this routing tree, we have three comparisons followed by a virtual comparison which then splits its weight among the three comparisons at the next level down in the routing tree. Let $\sigma' = \sigma/(2(3n)^2)$ be the weight of each comparison node at the root of the routing tree. The next node in the analysis tree (at global depth $i + \log(3n) + 2$) is the virtual node whose weight is

$$\sigma'/2 = \sigma/(4(3n)^2) = \sigma/(4^{\log(3n)+1}) \geq (1/4)^{i+\log(3n)+2}$$

The children of this node in the tree (at global depth $i + \log(3n) + 3$) each have weight

$$\begin{aligned}
\sigma'/(2 \cdot 2 \cdot 3) =& \sigma/(6 \cdot 4(3n)^2) \\
=& \sigma/(6 \cdot 4^{\log(3n)+1}) \\
\geq& \sigma/(4^2 \cdot 4^{\log(3n)+1}) \\
=& \sigma/(4^{\log(3n)+3}) \\
\geq& (1/4)^{i+\log(3n)+3}
\end{aligned}$$

We can map our analysis tree back on to the routing tree if we divide the weight by $2 \cdot 2 \cdot 3$ each level down the routing tree. This means the analysis tree has one more level beyond the last virtual comparison. This last level has three virtual comparisons per tree. This means that the number of levels in this routing tree is $2\log(\sqrt{n})+1$ and the weight at the bottom is the weight at the top divided by $(2 \cdot 2 \cdot 3)^{\log(\sqrt{n})}$ (here we need to use the number of levels in the routing tree). Thus we have the weight of each node at the bottom of the routing tree (at global depth $i + \log(3n) + 2\log(\sqrt{n}) + 1$) is

$$\begin{aligned}
\sigma/(2(3n)^2 \cdot (2 \cdot 2 \cdot 3)^{\log(\sqrt{n})}) \geq & \sigma/(4(3n)^2 \cdot (4 \cdot 3)^{\log(\sqrt{n})}) \\
\geq & \sigma/(4^{\log(3n)+1} \cdot (4 \cdot 3)^{\log(\sqrt{n})}) \\
\geq & \sigma/(4^{\log(3n)+1} \cdot (4 \cdot 4)^{\log(\sqrt{n})}) \\
\geq & \sigma/(4^{\log(3n)+1} \cdot (4^2)^{\log(\sqrt{n})}) \\
\geq & \sigma/(4^{\log(3n)+1} \cdot (4)^{2 \cdot \log(\sqrt{n})}) \\
= & \sigma/(4^{\log(3n)+2\log(\sqrt{n})+1}) \\
\geq & (1/4)^{i+\log(3n)+2\log(\sqrt{n})+1}
\end{aligned}$$

To finish the proof, note that the base case is realized in the very first call to the algorithm, since a comparison at depth 0 has weight $1 = (1/4)^0$. □

Lemma 11 (Cole 1987 **[21])** *For $j \geq 5(i + (1/2)\log(6n))$, during the $(j+1)^{st}$ iteration there are no active comparisons at depth i.*

Proof. At the start of the $(j+1)^{\text{st}}$ iteration the total active weight $\mathbb{W}$ is bounded by $(3/4)^{5(i+(1/2)\log(6n))} \cdot (3n)$ (by Lemma 9).

We note $(3/4)^5 < (1/4)$. So

$$\begin{aligned}
\mathbb{W} < & (1/4)^{i+(1/2)\log(6n)} \cdot (3n) \\
= & (1/4)^i \cdot (1/4)^{(1/2)\log(6n)} \cdot (3n) \\
= & (1/4)^i \cdot (1/(6n)) \cdot (3n) \\
= & (1/4)^i \cdot (1/2)
\end{aligned}$$

But an active comparison at depth i has weight at least $(1/4)^i$. So there is no such comparison. □

Goodrich and Pszona [28] point out that the dependency graph for one recursive call has $O(\log n)$ height, and one recursive call of boxsort performs $O(\log n)$ parallel steps. They also cite Motwani and Raghavan [44] stating that with high probability boxsort terminates in $O(\log n)$ parallel steps, and thus the height of the whole dependency graph of the parametric search boxsort also has height $O(\log n)$ with high probability. Plugging this height into Lemma 11 as the value for i, we get that we require $O(\log n)$ calls to the decision algorithm.

Theorem 4 *Given a chord $\ell \subset P_{in}$ we compute a CCOSKEG disc $D(c^*, \rho^*)$ in $O(n\log^2 n + m)$ time with high probability using $O(n\log r + m)$ space.*

Proof. Preprocessing from Section 2 takes $O(m)$ time and space. It will be shown in Appendix A.5 that with $O(\log n + \log r)$ calls to the decision algorithm and additional $O(n \log r)$ time and $O(n\log r + r)$ space, we compute the last reflex vertices on the paths from each point $u \in S$ to $\partial D(u, \rho^*) \cap \ell$, effectively giving us $O(n)$ items to sort. Given this result and Corollary 2, the preprocessing from Section 3.1 makes $O(\log n + \log r)$ calls to the decision algorithm of Lemma 3, uses $O(n \log r + r)$ space, and takes time

$$\begin{aligned}
& O(n\log r + \log n \cdot n(\log r + \log n) + \log r \cdot n(\log r + \log n)) \\
= & O(n \log n \log r + n\log^2 n + n\log^2 r)
\end{aligned}$$

As seen in Goodrich and Pszona [28], Motwani and Raghavan [44], and Reischuk [50], with high probability (i.e., at least $1 - e^{-\log^b n}$ for some constant $b > 0$) boxsort chooses a "good" sequence of pivots so that it only requires $O(\log n)$ calls to the decision algorithm of Lemma 3; and with the same probability, taking into account the number of recursive calls and the time we spend in a recursive call to create boxes and then sort the remaining comparisons into their boxes, using boxsort for parametric search takes $O(n\log n + \log n \cdot n(\log r + \log n)))$ time and $O(n + r)$ space.

Together with the preprocessing, the time to run our parametric search using boxsort is $O(n \log n \log r + n\log^2 n + n\log^2 r + m)$ with high probability and it uses $O(n \log r + m)$ space. It produces the CCOSKEG disc by the fact that the parametric search technique finds a SKEG disc for S centred on ℓ (i.e., a disc with minimum radius centred on ℓ containing at least k points of S).

Considering the $O(m)$ time spent preprocessing the polygon, we can simplify the runtime. Consider the largest-order terms in the running time:

$$\underbrace{n\log n\log r}_{A} + \underbrace{n\log^2 n}_{B} + \underbrace{n\log^2 r}_{C} + \underbrace{m}_{D} \tag{7}$$

Either $\log r < \log n$ or $\log n < \log r$, so A is always dominated by B or C. Consequently, Expression (7) can be simplified to

$$\underbrace{n\log^2 n}_{B} + \underbrace{n\log^2 r}_{C} + \underbrace{m}_{D} \tag{8}$$

Assume C dominates B and D. This implies:

$$
\begin{array}{llrr}
 & n\log^2 r \in \omega(m) & & (9)\\
 & n\log^2 r \in \omega(n\log^2 n) & & (10)\\
 & \log r \in \omega(\log n) & \text{by (10)} & (11)\\
\Rightarrow & r > n^3 & & \\
\Rightarrow & m > n^3 & & \\
\Rightarrow & m^{1/2} > n^{3/2} & & \\
\Rightarrow & m^{1/2} \in \Omega(n^{3/2}) & & \\
\Rightarrow & m^{1/2} \in \omega(n) & & (12)\\
 & m^{1/2} \in \omega(\log^2 m) & & \\
\Rightarrow & m^{1/2} \in \omega(\log^2 r) & & (13)\\
 & m \in \omega(n\log^2 r) & \text{by (12) and (13)} & (14)
\end{array}
$$

Consequently, Expression (8) can be simplified to $n\log^2 n + m$, meaning that the time to run our parametric search using boxsort is $O(n\log^2 n+m)$ with high probability. □

A.5 Decreasing to a Linear Number of Items to Sort

In this section, our goal is to discover, for each point of S, which reflex vertices to use for Eq. (1) when we have the optimal radius, giving us $O(n)$ equations / intersection points to use in the parametric search. We do so by using another parametric search. The procedure is straightforward; it is like one of the steps used in the boxsort parametric search of Goodrich and Pszona [28] presented in Appendix A.4, except here each comparison has one root. Similar to routing unmarked elements through the binary tree of sorted crossings, we independently route through $2n$ binary search trees of $O(\log r)$ height where the outcomes of the comparisons depend on the solution to the parametric search. This allows us to find the reflex vertices for each $u \in S$ that anchor $\partial D(u,\rho^*) \cap \ell$. However, instead of inferring the optimum by sorting intersection points described as equations from which candidates for the optimum are extracted, here our comparisons are directly in the parameter space: we are directly comparing distances against the optimum.

Since domain markers for the funnel of a point in S with ℓ are points along ℓ, in addition to defining domains for reflex vertices they also provide distances to use as candidate radii. We have the following monotonicity property: for any point $u \in S$, the distance to ℓ increases monotonically as we move from its closest point $u_c \in \ell$ to the endpoints of ℓ [48]. Thus, if we can decide how the radius produced by a given marker compares to the optimal radius, we can perform two binary searches among these markers between u_c and the endpoints of ℓ to find the domains which contain $\partial D(u,\rho^*) \cap \ell$, and hence discover which (at most two) reflex vertices to use in Eq. (1) for u in the main parametric search. Recall Observation 2 which uses the shortest-path data structure to extract the truncated funnel of u and ℓ and to perform a binary search along ℓ using the edges of this funnel.

We use this binary search to mimic the step of boxsort that routes the unmarked elements through the binary search tree of sorted crossings. The binary search can be represented as routing an element through a binary tree, discerning a particular path. Routing an element through these search trees is similar to following a directed path of $O(\log r)$ height. Each node on the path corresponds to a comparison that must be resolved before the routing is able to continue on to the next node in the path. As we route along these paths, the current node in a path is the *active comparison* in the path. When we have enough information (i.e., how ρ^* relates to the candidate at this comparison), we resolve the comparison and it is no longer active. A dependence relation arises wherein a node cannot be active until all its ancestors have been resolved, at which point it may be immediately resolved and inactivated if it is known how ρ^* relates to the candidate radius of the current comparison node; otherwise, the comparison remains active until some call to the decision algorithm reveals the relation of ρ^* to the candidate.

Since we know u_c and we have the monotonicity property, once a comparison is resolved then we know for an extension segment e in constant time and space to which side of $\ell \cap e$ a point along ℓ of distance ρ^* to u lies. Ignoring the calls to the decision algorithm, the time spent over all of the points of S to discover which reflex vertices to use for Eq. (1) in the main parametric search (i.e., the time to perform the binary searches along ℓ, or, equivalently, route through the binary trees) is $O(n\log r)$. The space is $O(r + n\log r)$ due to the fact that we will be holding on to all $2n$ shortest-path query structures at once to sequentially simulate a parallel algorithm, and since careful reading of Guibas and Hershberger [23, 29, 34] implies that a truncated funnel takes up $O(\log r)$ extra space. Each point of S has up to two independent binary trees of $O(\log r)$ height through which it is routing to find the domain of interest.

First, for each $u \in S$ and the endpoints ℓ_1 and ℓ_2 of ℓ, we compute the truncated funnel between u, ℓ_1 and u_c; and the truncated funnel between u, u_c, and ℓ_2. This is done in $O(n\log r)$ time and $O(r + n\log r)$ space via Observation 1 and the space analysis in the previous paragraph. Either of these funnels can replace the one from Observation 2 to yield the same time and space for the searches.

Then we sequentialize the running of $2n$ parallel searches through binary trees of $O(\log r)$ height (one per funnel). For each point $u \in S$ we search through the domain markers implicitly contained in its two funnels looking for the domains that contain $\partial D(u, \rho^*) \cap \ell$, which are the domains that contain a point that is distance ρ^* away from u.

For each tree through which we are routing, each step produces a comparison to resolve. Since a call to the decision algorithm is considered costly, we do not want to call the decision algorithm to resolve each comparison individually. Using the fact that the candidate radii have the monotonicity property we need for parametric search (i.e., given the relation between ρ^* and a candidate radius, we know the relation between ρ^* and either everything bigger than or less than the candidate) and the fact that the domain markers used in the comparisons of the searches are also candidate radii, we can route through the trees with a logarithmic number of calls to the decision algorithm. Following Goodrich and Pszona [28], we assign weights to the comparisons in the searches. Initially, each gets a weight of 1; when a comparison is resolved its child receives half of its weight. The sum of the weights of the currently active candidates is called the *active weight*. The routing can be considered as iterations involving three steps: in the first step, we use a linear-time weighted-median-finding algorithm [51] to choose the weighted median candidate radius (as described in Appendix A.4); in the next step, we input that radius into the decision algorithm; when the decision algorithm returns, the last step is to repeatedly resolve all active comparisons that can be resolved until no more routing can be performed without knowing the result of another call to the decision algorithm. At this point, the next iteration begins.

We can follow the approach of Goodrich and Pszona [28] and use Cole's analysis [21], which we repeat here modified for this specific case, to show that there are $O(\log n + \log r)$ calls to the decision algorithm.

Lemma 12 (Cole 1987 [21]) *At the start of the $(j+1)^{st}$ iteration, the active weight is bounded above by $(3/4)^j \cdot (2n)$ for $j \geq 0$.*

The proof of Lemma 12 is similar to that of Lemma 9 and is omitted.

Lemma 13 (Goodrich and Pszona 2013 [28]) *Each comparison at depth i has weight $\geq (1/4)^i$.*

Proof. The first comparison node in each of these $2n$ paths representing the search, i.e., depth 0, has weight $1 \geq (1/4)^0$. Each step down the path halves the weight of its parent, so at depth i the weight is $(1/2)^i \geq (1/4)^i$. □

Lemma 14 (Cole 1987 [21], Goodrich and Pszona 2013 [28]) *For $j \geq 5(i + (1/2)\log(4n))$, during the $(j+1)^{st}$ iteration there are no active comparisons at depth i.*

Proof. At the start of the $(j+1)^{\text{st}}$ iteration the total active weight $\mathbb{W}$ is bounded by $(3/4)^{5(i+(1/2)\log(4n))} \cdot (2n)$ (by Lemma 12).

We note $(3/4)^5 < (1/4)$. So $\mathbb{W} < (1/4)^{i+(1/2)\log(4n)} \cdot (2n) = (1/4)^i \cdot (1/4)^{(1/2)\log(4n)} \cdot (2n) = (1/4)^i \cdot (1/(4n)) \cdot (2n) = (1/4)^i \cdot (1/2)$. But an active comparison at depth i has weight at least $(1/4)^i$. So there is no such comparison. □

Plugging the height of the binary search trees into Lemma 5 as i, we get the following.

Corollary 15 *With $O(\log n + \log r)$ calls to the decision algorithm, with additional $O(n \log r)$ time and $O(r + n \log r)$ space, we compute for each $u \in S$ the anchors of $\partial D(u, \rho^*) \cap \ell$.*

B Depth Bounds

Consider the following. Imagine we preprocess P_{in} by simplifying it to P and then triangulating[8] P. If it is known that the points of S in a SKEG disc lie completely in one of those triangles, then we can solve the SKEG problem as follows. First we build Kirkpatrick's [18, 36] $O(\log r)$ query-time point-location data structure on these triangles in $O(r)$ time with $O(r)$ space. For each of the $O(r)$ triangles we build a list of the points of S contained within. So far we have used $O(n \log r + m)$ time and $O(n + m)$ space.

Now we iterate over the triangles and in each triangle use an exact algorithm for the smallest k-enclosing disc for planar instances [32]. If triangle i has n_i points of S in it, then we run the exact algorithm in $O(n_i k)$ expected time and $O(n_i + k^2)$ expected space. We know over all of the triangles, the n_i sum to n, so we have the following.

Theorem 16 *Simplify P_{in} to P and triangulate P. If the points of S in a SKEG disc are contained in a triangle of P, we solve the SKEG disc problem in $O(n \log r + nk + m)$ expected time and $O(n + k^2 + m)$ expected space.*

Below we show some bounds on the *depth* of a geodesic disc whose radius is at most four times the optimal radius of a SKEG disc, ρ^*. The depth of a disc is the number of points of S contained within. We assume no four points are geodesically co-circular, thus there are at most k points in any disc of the optimal radius ρ^*. In this paper, the $depth(\rho)$ is the maximum depth over all points in the polygon P for a geodesic disc of radius ρ. By definition, $depth(\rho^*) = k$. In this subsection, we assume that $n > kr$ (otherwise the bounds should be expressed as $\min(n, kr)$).

Lemma 17 *We have $depth(2\rho^*) \in \Omega(kr)$ under our general position assumption.*

Proof. Consider the optimal disc. It could jut into $\Omega(r)$ spikes of the polygon (i.e., contain $\Omega(r)$ reflex vertices, each of which obstructs visibility between points in the disc; e.g., Fig. 1). In each spike, if we put a disc of radius ρ^* on the boundary of the optimal disc, it could contain $\Omega(k)$ points. Then a disc of radius $2\rho^*$ centred on the optimal disc's centre has $\Omega(kr)$ points in it. □

Lemma 18 *For a constant $c > 1$, we have $depth(c\rho^*) \in \Theta(kr)$ under our general position assumption.*

[8]Note that building the shortest-path data structure of Guibas and Hershberger [29, 34] triangulates P, as does Kirkpatrick's [18, 36] point-location data structure. They both run in linear time since we have linear-time polygon triangulation algorithms [7, 18].

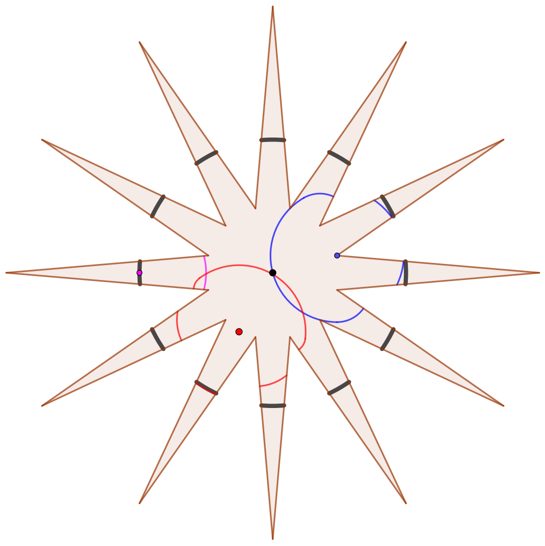


Figure 1: A star-shaped simple polygon with a geodesic disc of radius 1 and some attempts to cover multiple spikes with geodesic discs of radius 0.5.

Proof. The lower bound from Lemma 17 can be directly extended to a radius of $c\rho^*$.

We now show the upper bound. Consider the geodesic disc of radius $c\rho^*$ centred on $u \in P$, $D(u, c\rho^*)$. Let $T(P, u)$ be the shortest path tree of u and let $E(P, u)$ be the set of extension segments of $T(P, u)$. Consider the tree $T(P, u) \cup E(P, u)$. The tree $T(P, u) \cup E(P, u)$ subdivides P into $O(r)$ Euclidean triangles such that every point q in the triangle has the same anchor on $\Pi(u, q)$ [9, Note 3.10][30]. Thus, $D(u, c\rho^*)$ may intersect $O(r)$ triangles. Within any given triangle Δ, any portion of a geodesic disc appears Euclidean. As such, $D(u, c\rho^*) \cap \Delta$ (which looks locally in Δ like a Euclidean disc of radius at most $c\rho^*$) can be covered by a constant number of discs of radius ρ^* (due to the bounded doubling dimension of the Euclidean metric), each with at most k points of S in it. The bound follows. □

These bounds hold in general if nothing more is known about the manner in which a 2-approximation is produced. However, if we combine the CCOSKEG disc algorithm from Section 3 with something similar to Theorem 16 from the beginning of the section to produce a 2-approximation, then we can get a better upper bound on the number of points of S in a disc with that radius.

We assume the preprocessing of Section 2 and the preprocessing for Theorem 16 (including building a list of the points of S in each triangle) has been performed in $O(n \log r + m)$ time and $O(n + m)$ space. Either the points of S of a SKEG disc lie completely in a triangle of P, or the disc contains a point of S from each side of some diagonal. If the points of the disc are contained in a triangle, then running the expected linear-time 2-approximation algorithm for planar instances [32] in each triangle will give us a 2-approximation. Making the appropriate change in Theorem 16, we have spent $O(n \log r + m)$ expected time and $O(n + m)$ expected space. If a SKEG disc contains at least one point of S from each side of some diagonal, then running the algorithm of Theorem 4 from Section 3.3 on each diagonal will give us a 2-approximation. Either a SKEG disc is centred on a diagonal, in which case we find it; or a SKEG disc intersects a diagonal, in which case when processing that diagonal we either compute a KEG disc centred on a point inside that SKEG disc, or a KEG disc centred on a point outside of the SKEG disc that gives us a KEG disc with a smaller radius than any KEG disc centred on a point of the diagonal inside the SKEG disc, either of which gives us a 2-approximation. Running the CCOSKEG disc algorithm of Theorem 4 on each diagonal, we spend $O(nr \log^2 n + nr \log^2 r + m)$ expected time[9] and use $O(n \log r + m)$ space.

Thus, in $O(nr \log^2 n + nr \log^2 r + m)$ expected time we have produced a 2-approximation using $O(n \log r + m)$ expected space. Let the 2-approximation radius we have found be ρ_2. We now prove $depth(\rho_2) \leq 10k$. Either a disc of radius ρ_2 is centred on a diagonal, or in a triangle of P. By definition, any disc of radius ρ_2 centred on a diagonal has at most k points in it. Now consider a disc D_2 of radius ρ_2 centred in a triangle of P. The boundary of this disc could intersect the three diagonals of the triangle. Consider the closest point of one of the diagonals to the centre of D_2, and create a disc D_ℓ of radius ρ_2 centred there. The portion of D_2 on the other side of the diagonal is contained in D_ℓ, and thus contains at most k points on the other side of the diagonal since ρ_2 is at most the radius of the CCOSKEG disc on this diagonal. Thus, at most $3k$ points of S in D_2 come from outside the triangle. Inside the triangle, locally it looks like the Euclidean plane. Thus, by the bounded doubling dimension of the Euclidean plane, D_2 contains at most $7k$ points of S contained in the triangle. If instead of *modifying* Theorem 16 we use the exact algorithm it specifies in each triangle, then the portion of D_2 inside the triangle captures at most k points of S, in which case we get $depth(\rho_2) \leq 4k$.

Theorem 19 *In $O(nr \log^2 n + nr \log^2 r + m)$ expected time we compute a radius ρ_2 using $O(n \log r + m)$ expected space that is a 2-approximation to ρ^* such that $depth(\rho_2) \leq 10k$. If we use $O(nr \log^2 n + nr \log^2 r + nk + m)$ expected time and $O(n \log r + k^2 + m)$ expected space, we can improve ρ_2 such that $depth(\rho_2) \leq 4k$.*

[9]The runtime stated in Theorem 4 hides a term dominated by the preprocessing time. Since we do not run the preprocessing for each diagonal, that simplification does not apply to this expression.

C Figures

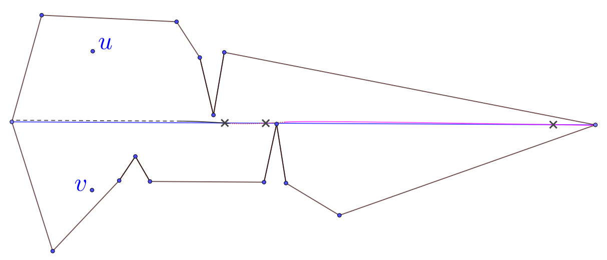


Figure 2: The bisector of points u and v on opposite sides of the blue chord of the polygon can intersect the chord $\Theta(r)$ times. The intersections are labelled with "$\times$". The different arcs that form the bisector are drawn with different ink styles (e.g., dashed vs dotted vs normal ink).

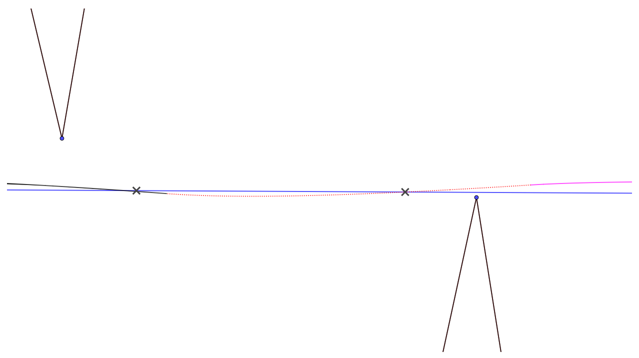


Figure 3: Zoomed-in view of the first two crossings of Fig. 2.

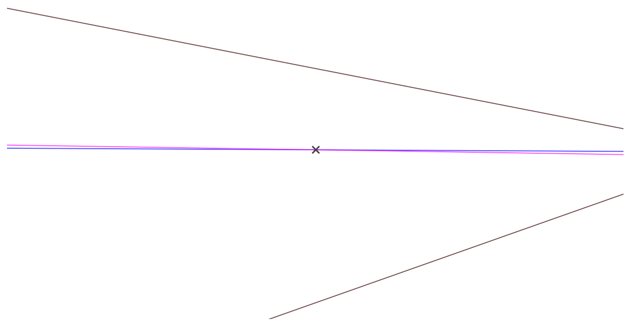


Figure 4: Zoomed-in view of the third crossing of Fig. 2. As one zooms in infinitesimally and the right side of the polygon moves toward infinity, more reflex vertices can be added to force the bisector to cross $\Theta(r)$ times.

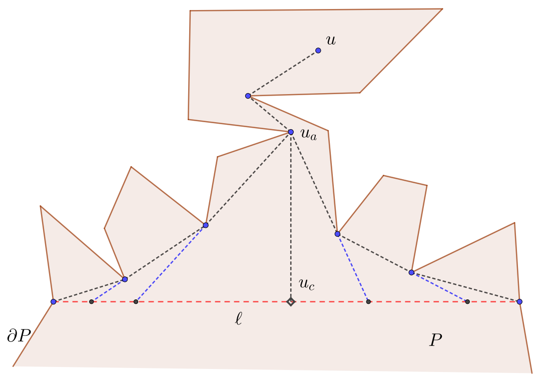


Figure 5: The funnel from u to the endpoints of ℓ, including the apex u_a and the projection u_c of u onto ℓ. Also seen are the extensions of funnel edges (in blue) and their intersection points with ℓ. These intersection points can be used to perform a binary search along ℓ.

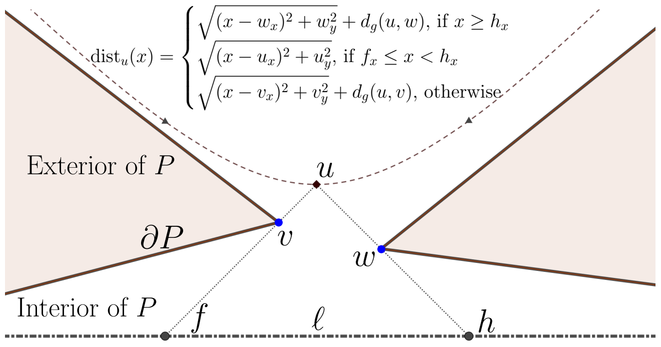

$\mathrm{dist}_u(x) = \begin{cases} \sqrt{(x-w_x)^2+w_y^2} + d_g(u,w), \text{ if } x \geq h_x \\ \sqrt{(x-u_x)^2+u_y^2}, \text{ if } f_x \leq x < h_x \\ \sqrt{(x-v_x)^2+v_y^2} + d_g(u,v), \text{ otherwise} \end{cases}$
Exterior of P
u
∂P
v
w
Interior of P
f
ℓ
h

Figure 6: Considering the chord ℓ of P to be the x-axis, given a point $u \in S$ we refer to the dashed graph of the function $\mathrm{dist}_u(\cdot)$ as the distance function of u to ℓ. The points f and h on ℓ mark where different pieces of $\mathrm{dist}_u(\cdot)$ begin.

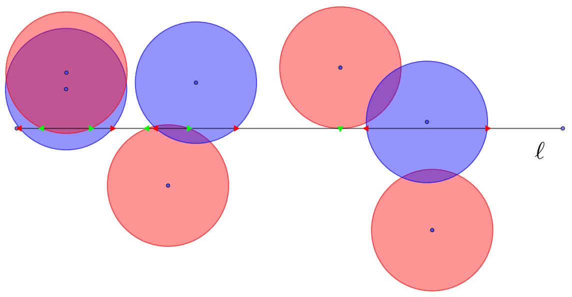


Figure 7: The geodesic discs (arbitrarily red and blue) of radius ρ^* centred on points of S (blue points) intersect ℓ. The intersection points of red (blue) disc boundaries with ℓ are marked by green (red) triangles.

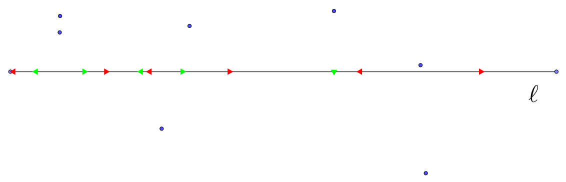


Figure 8: We use parametric search to sort the intersection points of disc boundaries (i.e., the red and green triangles) from Fig. 7 along ℓ, without knowing ρ^*, and are able to deduce ρ^* in the process.

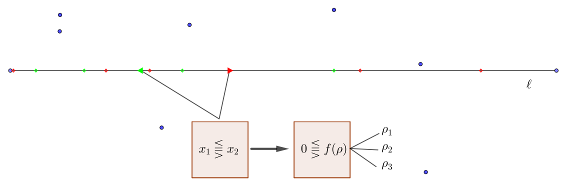


Figure 9: Parametric search lets us discover the relative order of two endpoints (e.g., the green and red triangles at positions x_1 and x_2 respectively) along ℓ. A resolved comparison in the generic algorithm reveals which of the two intersection points is to the left of the other when using ρ^* as the disc radii. Before a comparison can be resolved, the algorithm must solve for the roots of the two intersection equations (at most three roots in our case).

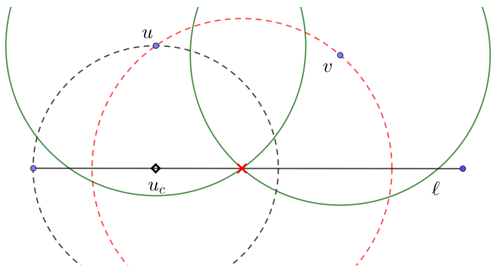


Figure 10: The CCOSKEG disc may be defined by one point on its boundary (such as the black dashed disc centred on the black hollow diamond representing u_c), in which case ρ^* is the distance from some point $u \in S$ to its projection u_c; or it is defined by at least two points, e.g., $u, v \in S$, on its boundary (such as the red dashed disc centred on the red "$\times$"), in which case ρ^* is the radius such that the intersection of the boundaries of the green discs centred at u and v, i.e., $\partial D(u, \rho^*) \cap \partial D(v, \rho^*)$, is the red "$\times$".

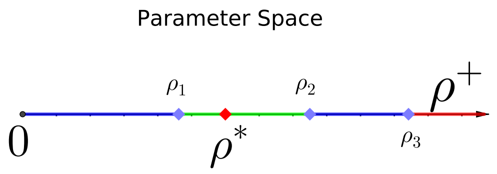


Figure 11: A comparison of intersection points along ℓ produces at most three roots and defines at most four intervals in the parameter space. Due to the monotonicity property of the parameter we are trying to optimize (i.e., radius of a CCOSKEG disc), once we determine which interval contains ρ^* we can resolve the comparison that produced the roots.

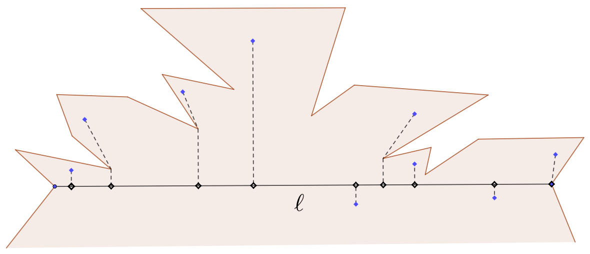
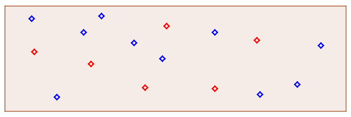
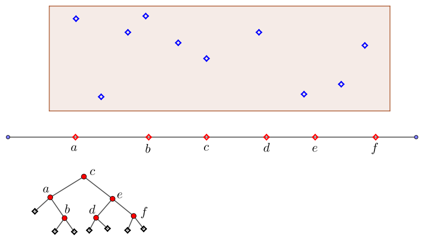


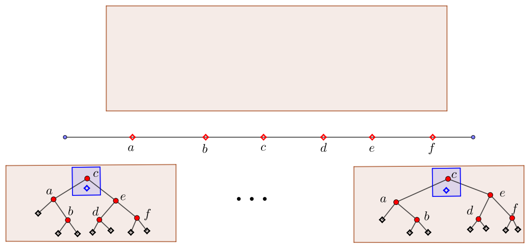
Figure 12: An example of points of S (blue diamonds) and their projections onto ℓ (hollow black diamonds).



(a) We begin with a collection of equations (depicted as diamonds in a box) representing the intersection points of the disc boundaries with ℓ. Going forward we no longer distinguish between the equations and the points they represent. Of the n points, we randomly select $\sqrt{n}$ points to act as pivots. The selected points are red, and the rest are blue.



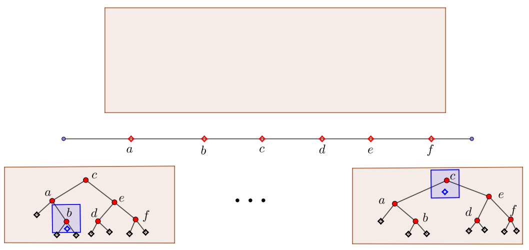
(b) We then decide the relative order of the red pivots along ℓ (i.e., decide their relative ordering) by creating $O(n)$ comparisons between them and using a logarithmic number of calls to the decision algorithm to resolve them. In this figure, the sorted red points are labelled a through f and carve ℓ up into seven relatively sorted intervals / boxes into which we must place the remaining blue points. This can be done by performing a binary search on the red pivots. The red pivots can be considered as creating a binary search tree (with c as the root in this example).



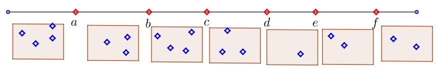
(c) The routing of each blue point through the tree of red pivots can be done independently of the other blue points, allowing us to perform their routing in parallel. For a blue point we are routing through the tree, each step in the tree creates a comparison between the blue point and the red pivot represented by the tree node. To route the elements through the tree, we repeatedly select the weighted median of the roots produced by the available comparisons and use this median in a call to the decision algorithm. After the call to the decision algorithm, we resolve whichever comparisons it is possible to resolve (which moves a blue point down the tree), and repeat until it is known into which intervals along ℓ each blue point belongs.

Figure 13: An illustration of a recursive call for boxsort when used as the sorting algorithm in parametric search.

Figure 13: An illustration of a recursive call for boxsort when used as the sorting algorithm in parametric search.

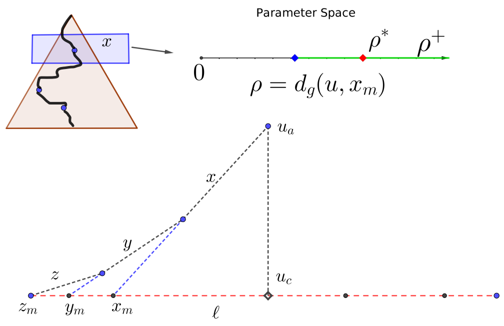


(d) Since the decision algorithm is called on the weighted median of the candidates and each instance of routing through the trees is independent, at any point in this routing process blue points may be at different levels in the binary search trees of red pivots.



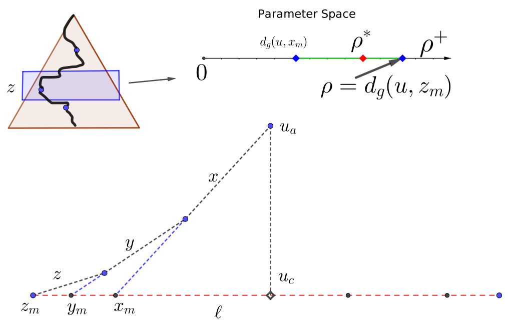
(e) Once it is known into which interval each blue point belongs, we collect them together in each interval to begin the next recursive calls.

Figure 13: An illustration of a recursive call for boxsort when used as the sorting algorithm in parametric search.

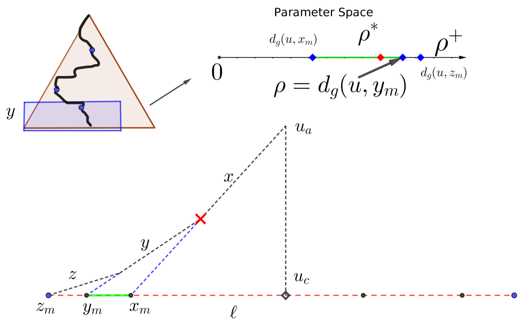


(a) Shown is the funnel between u_a for a point $u \in S$, its projection u_c, and an endpoint of ℓ. The convex chain representing the funnel edges is stored in a binary search tree (which stores the edges). In this example, the funnel edges from u_a to the left endpoint of ℓ are x, y, and z, and their extension segments (the blue dashed edges) intersect ℓ at the markers x_m, y_m, and z_m respectively. We can use the distances between u and the markers as candidate radii in the decision algorithm. Since the distance from u to the points on ℓ increases monotonically as we move from u_c to the left endpoint of ℓ, we can use these distances to find an interval in the parameter space in which ρ^* lies, and at the same time the interval along ℓ between two markers where the disc of radius ρ^* centred at u intersects ℓ. In this example, the decision algorithm has determined that the optimal radius is larger than $d_g(u, x_m)$, so we continue down the binary search tree on the side that brings us closer to the left endpoint of ℓ.

Figure 14: An illustration of the method presented in Section 3.4 for determining the reflex vertices to use in Eq. (1).

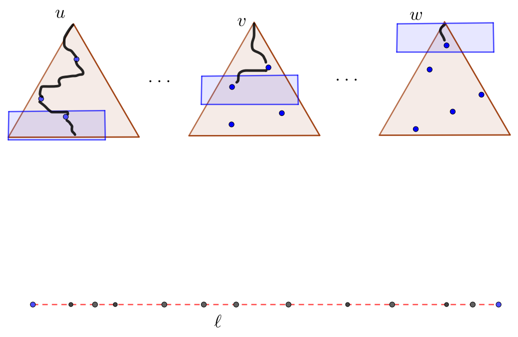


(b) After the edge x we visited the edge z. The decision algorithm has determined that the optimal radius is less than $d_g(u, z_m)$, so we continue down the binary search tree on the side that brings us closer to x_m.



(c) After the edge z we visited the edge y. The decision algorithm has determined that the optimal radius is less than $d_g(u, y_m)$, so we conclude that $D(u, \rho^*)$ intersects ℓ in the green interval between y_m and x_m, and the reflex vertex to use in Eq. (1) is the one marked by the red $\times$.

Figure 14: An illustration of the method presented in Section 3.4 for determining the reflex vertices to use in Eq. (1).



(d) Similar to boxsort (see Fig. 13), routing through the funnels to find the required reflex vertices is done in parallel, repeatedly testing the radius produced by the median weighted comparison with the decision algorithm and then resolving any number of comparisons. The routing process for different funnels may be at different depths in the trees at any given moment. In this example, the routing for $u \in S$ has finished, the routing for $v \in S$ is halfway through its binary search, and the routing for $w \in S$ is still at the beginning.

Figure 14: An illustration of the method presented in Section 3.4 for determining the reflex vertices to use in Eq. (1).

Parallel Line Centers with Guaranteed Separation[*]

Chaeyoon Chung[†] Taehoon Ahn[*] Sang Won Bae[‡] Hee-Kap Ahn[§]

Abstract

Given a set P of n points in the plane and an integer $k \geq 1$, the k-line-center problem asks k slabs whose union encloses P that minimizes the maximum width of the k slabs. In this paper, we introduce a new variant of the k-line-center problem for $k \geq 2$, in which the resulting k lines are parallel and a prescribed separation between two line centers is guaranteed. More precisely, we define a measure of separation, namely the gap-ratio of k parallel slabs, to be the minimum distance between any two slabs, divided by the width of the smallest slab enclosing the k slabs. We present the first and efficient algorithms for the following problems: (1) Given a real $0 < \rho \leq 1$, compute k parallel slabs of minimum width that cover P with gap-ratio at least ρ. (2) Compute k parallel slabs that covers P with maximum possible gap-ratio. Our algorithms run in $O(\rho^{-k} \cdot kn \log n)$ and $O(\rho_{\max}^{-k} \cdot kn \log n)$ time, respectively, where $\rho_{\max}$ denotes the maximum possible gap-ratio of any k parallel slabs that cover P. Both algorithms use $O(n)$ space.

1 Introduction

In many practical situations of geometric facility location, one would like to locate two or more facilities of a certain shape that serve a given set P of input customers (e.g., points) in the plane, in such a way that the interference between two facilities and/or between two customers served by different facilities is minimized. Hence, most preferred in this case are mutually disjoint facilities with separation guaranteed or maximized between the covering regions of any two facilities. In this

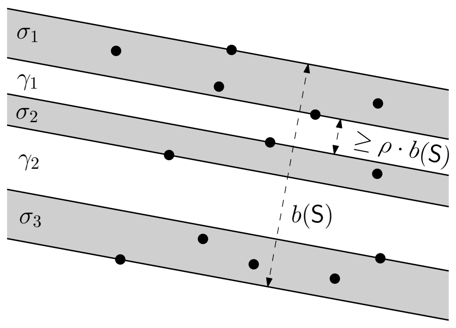


Figure 1: An example of a 3-slab cover S of 13 points in P. Here, we have $w(\mathsf{S}) = w(\sigma_1) = w(\sigma_3)$ and $g(\mathsf{S}) = w(\gamma_1) \geq \rho \cdot b(\mathsf{S})$. Hence, the k-slab S is a minimum-width k-slab cover of P whose gap-ratio is at least ρ.

paper, we consider such a variant of the *k-line-center problem* in the plane for $k \geq 2$ with separation guaranteed or maximized between any two line centers and between their covering regions. By this goal regarding separation, it is obvious that the resulting k line centers are parallel and mutually disjoint. A more precise description of our problem is given below.

As an input point is assigned to its closest line center under the Euclidean metric, the covering region of a line center forms a *slab*. So the problem is equivalent to finding k parallel and mutually disjoint slabs $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that their union encloses P and $\sigma_i \cap P \neq \emptyset$ for each $i = 1, \ldots, k$. We call a sequence S of such k parallel slabs a *k-slab cover* of P, or just a *k-slab* regardless of the relation to P. For each slab σ, its *width* $w(\sigma)$ is the orthogonal distance between its two bounding lines. Consider a k-slab $\mathsf{S} = (\sigma_1, \ldots, \sigma_k)$. (See Figure 1.) The *width* of S, denoted by $w(\mathsf{S})$, is defined to be $\max\{w(\sigma_1), \ldots, w(\sigma_k)\}$, and the *breadth* of S, denoted by $b(\mathsf{S})$, is the orthogonal distance between the two outermost bounding lines of S. The region between two consecutive slabs σ_i and σ_{i+1} in S is called a *gap*, denoted by γ_i. Each gap γ_i also forms a slab, so $w(\gamma_i)$ denotes its width. The *gap-width* of S is defined to be the minimum of $w(\gamma_i)$ over $i = 1, \ldots, k-1$, denoted by $g(\mathsf{S})$. Our measure of separation in the problem is then defined to be the ratio of the gap-width over the breadth, namely, the *gap-ratio* of S is $\rho(\mathsf{S}) := g(\mathsf{S})/b(\mathsf{S})$.

The goal of the problem is thus to find an optimal k-slab cover of P regarding two objective criteria: width and gap-ratio. More precisely, we consider the following

[*]C.Chung, T.Ahn, and H.-K.Ahn were 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 (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)). S.W.Bae was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2018R1D1A1B07042755 and No. RS-2023-00251168).

[†]Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Republic of Korea, {chaeyoon17, sloth}@postech.ac.kr

[‡]Division of Artificial Intelligence and Computer Science, Kyonggi University, Suwon, Republic of Korea, swbae@kgu.ac.kr

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

problems for given $k \geq 2$ and a set P of n points:

1. For a given real $0 < \rho \leq 1$, find a minimum-width k-slab cover of P whose gap-ratio is at least ρ.

2. Find a maximum-gap-ratio k-slab cover of P.

Related work. The *k-line-center* problem is equivalent to finding k slabs, being not necessarily parallel, of min-max width whose union encloses the input points. The 1-line-center problem can be solved in $O(n \log n)$ time by computing the center line of a minimum-width slab of the points [9, 11]. For the 2-line-center problem, Agarwal and Sharir [3] gave an $O(n^2 \log^5 n)$-time algorithm, and Jaromczyk and Kowaluk [12] improved it to $O(n^2 \log^2 n)$ time. Agarwal et al. [1] gave an $(1+\epsilon)$-approximation algorithm for the problem that runs in $O(n(\log n + \epsilon^{-2}\log(1/\epsilon)) + \epsilon^{-7/2}\log(1/\epsilon))$ time. When k is part of input, the k-line-center problem is NP-complete because it is known to be NP-complete to decide whether n points can be covered by k lines [13]. Agarwal et al. [2] gave an $(1+\epsilon)$-approximation algorithm that runs in $O(n \log n)$ time with the constant factor depending on k and ϵ.

If the line centers are constrained to be parallel in the k-line center problem, the problem is equivalent to finding a minimum-width k-slab cover in our sense. In this case, Bae [5] presented an $O(n^2)$-time algorithm for the case of $k = 2$. No non-trivial algorithm is known for each $k > 3$, to the best of our knowledge.

Our results and approach. We present efficient algorithms for the above two problems. Our algorithms run in $O(\rho^{-k} \cdot kn \log n)$ and $O(\rho_{\max}^{-k} \cdot kn \log n)$ time, respectively, where $\rho_{\max}$ denotes the maximum possible gap-ratio of any k parallel slabs that cover P. Both algorithms use $O(n)$ space.

We first consider Problem 1 of computing a minimum-width k-slab cover of P whose gap-ratio is at least a given parameter $\rho \in (0, 1]$. To tackle the problem, we introduce a concept of a *separator* of a k-slab, defined to be a sequence of $k-1$ points each of which lies in its distinct gap. We describe efficient algorithms to compute an optimal k-slab cover that respects a fixed separator in Section 3. Then, in Section 4, we show how to solve Problem 1 by testing $O(1/\rho^k)$ candidate separators. In Section 5, we show that the algorithms in Sections 3 and 4 are helpful enough to achieve an efficient algorithm for Problem 2.

2 Preliminaries

The line segment connecting two points p and q is denote by pq and its length by $|pq|$. The *orientation* of a line ℓ in the plane is the angle swept from a vertical line in counterclockwise direction to ℓ. Hence, the orientation θ of each line falls in the range $\theta \in [0, \pi)$. The *orientation* of a linear object, including line segments, slabs, and k-slabs, is that of a line parallel to it.

For any two points p, q and orientation $\theta \in [0, \pi)$, define $d_\theta(p, q)$ to be the orthogonal distance between two lines in orientation θ through p and q. It can be written $d_\theta(p, q) = |pq| \cdot |\sin(\theta - \theta_{pq})|$, where θ_{pq} is the orientation of pq. Thus, $d_\theta(p, q)$ for fixed p and q is a sinusoidal function in θ. A function is called *sinusoidal* if it is of the form $a \sin(\theta + b)$ for some constants $a, b \in \mathbb{R}$.

Consider any non-vertical k-slab $\mathsf{S} = (\sigma_1, \ldots, \sigma_k)$ such that σ_i lies above σ_{i+1} and its i-th gap γ_i lies between σ_i and σ_{i+1} for $i = 1, \ldots, k-1$. We call S a (k, ρ)-*slab* if its gap-ratio $\rho(\mathsf{S}) \geq \rho$ for a constant $0 < \rho \leq 1$. Let $\mathsf{R} = (r_1, \ldots, r_{k-1})$ be a sequence of $k-1$ points on a common line in the order along the line. If it holds $r_i \in \gamma_i$ for each $i = 1, \ldots, k-1$, we call R a *separator* of S and say that the k-slab S respects the separator R.

3 (k, ρ)-Slabs Respecting a Given Separator

In this section, we present two algorithms that compute a minimum-width (k, ρ)-slab cover S^* of P respecting a given separator R. Since the k slabs in a k-slab cover of P are mutually disjoint and each of them encloses at least one point of P, we can assume that $0 < \rho \leq 1/(k-1)$. There is no k-slab cover of P for $\rho > 1/(k-1)$.

Let $\mathsf{R} = (r_1, \ldots, r_{k-1})$ be a given separator. Without loss of generality, we assume that the points $r_1, \ldots, r_{k-1}$ in R lie on a common vertical line in this order along it downwards. For each orientation $\theta \in [0, \pi)$, let $\ell_i(\theta)$ be the line in orientation θ through r_i for $i = 1, \ldots, k-1$. Then, these $k-1$ lines partition P into k disjoint subsets $P_1(\theta), \ldots, P_k(\theta)$ such that $P_1(\theta)$ consists of all points in P lying on or above $\ell_1(\theta)$, $P_i(\theta)$ for $1 < i \leq k-1$ consists all points in P lying on or above $\ell_i(\theta)$ and below $\ell_{i-1}(\theta)$, and $P_k(\theta)$ consists of the rest that lies below $\ell_{k-1}(\theta)$.

For $i = 1, \ldots, k$, let $\sigma_i(\theta)$ be the minimum-width slab in orientation θ that encloses $P_i(\theta)$. Let $\mathsf{S}(\theta) := (\sigma_1(\theta), \ldots, \sigma_k(\theta))$ be the k-slab cover of P consisting of these k parallel slabs. Let $\gamma_1(\theta), \ldots, \gamma_{k-1}(\theta)$ be the gaps of $\mathsf{S}(\theta)$. Note that $\mathsf{S}(\theta)$ respects the given separator R by definition. Thus, our goal is to find an optimal orientation $\theta^* \in (0, \pi)$ that minimizes the width of $\mathsf{S}(\theta)$ for all θ such that the gap-ratio of $\mathsf{S}(\theta)$ is at least the given threshold ρ.

By an abuse of notations, we let $w_i(\theta) = w(\sigma_i(\theta))$ for $i = 1, \ldots, k$ and $g_i(\theta) = w(\gamma_i(\theta))$ for $i = 1, \ldots, k-1$. Also, we let $w(\theta) = w(\mathsf{S}(\theta))$, $g(\theta) = g(\mathsf{S}(\theta))$, $b(\theta) = b(\mathsf{S}(\theta))$, and $\rho(\theta) = \rho(\mathsf{S}(\theta))$. By definition, note that $w(\theta) = \max w_i(\theta)$, $g(\theta) = \min g_i(\theta)$, and $\rho(\theta) = g(\theta)/b(\theta)$.

For each $\theta \in (0, \pi)$ and $1 \leq i \leq k$, we denote by $q_i^+(\theta)$ and $q_i^-(\theta)$ the two points of $P_i(\theta)$ such that every point of $P_i(\theta)$ lies in between the two lines ℓ^+ and ℓ^- in orientation θ passing through $q_i^+(\theta)$ and $q_i^-(\theta)$, respectively, and ℓ^+ lies above ℓ^-. We call $q_i^+(\theta)$ and $q_i^-(\theta)$ the *extreme* points of $P_i(\theta)$. Observe that the i-th slab $\sigma_i(\theta)$ is determined by the two lines in orientation θ through $q_i^+(\theta)$ and $q_i^-(\theta)$, and the i-th gap $\gamma_i(\theta)$ by the two lines in orientation θ through $q_i^-(\theta)$ and $q_{i+1}^+(\theta)$. This implies that their widths $w_i(\theta)$ and $g_i(\theta)$ are *sinusoidal* functions over a subdomain in which $q_i^+(\theta)$, $q_i^-(\theta)$, and $q_{i+1}^+(\theta)$ remain the same [5].

Note that if $P_i(\theta)$ consists of a single point p, we have $q_i^+(\theta) = q_i^-(\theta) = p$ and $w_i(\theta) = 0$.

If $P_i(\theta) = \emptyset$, $q_i^+(\theta)$ and $q_i^-(\theta)$ are not defined. Since the i-th slab σ_i contains no point of P, the k-slab $\mathsf{S}(\theta)$ is not defined. Thus, in this case, we set $w_i(\theta) = 0$ and $g_{i-1}(\theta) = g_i(\theta) = 0$ so that $g(\theta) = \rho(\theta) = 0$, and thus our algorithms filter out such orientations $\theta \in [0, \pi)$ in searching for an optimal (k, ρ)-slab cover of P.

3.1 First algorithm using $O(kn)$ space

In the following lemma, we show that each of these width functions is indeed piecewise sinusoidal and can be specified in an efficient way. This can be done by applying a geometric dualization [8, Chapter 8], the Zone Theorem [7] in the arrangement of lines, and efficient algorithms to compute the zone of a line in the arrangement [4, 15].

Lemma 1 *Each of the functions $w_1, \ldots, w_k$, $g_1, \ldots, g_{k-1}$, and b is piecewise sinusoidal with $O(n)$ breakpoints over the domain $[0, \pi)$, and an explicit description of each can be computed in $O(n \log n)$ time.*

The proof of Lemma 1 can be found in the full version.

By Lemma 1, we can also specify the functions w and g. Recall that w is the upper envelope of the w_i's and g is the lower envelope of the g_i's. Hence, we observe that w and g are piecewise sinusoidal with $O(kn)$ breakpoints, and can be computed explicitly in $O(kn \log k)$ time by a merge-sort-like recursive merging on k (or $k-1$) functions of $O(n)$ complexity. In conclusion we have the following algorithm.

Lemma 2 *Given P, $k \geq 2$, $0 < \rho \leq 1$, and a separator R as defined above, a minimum-width (k, ρ)-slab cover of P respecting R can be computed in $O(kn \log n)$ time and $O(kn)$ space, if exists.*

Proof. Here, we describe our algorithm. First, we compute the full descriptions of functions w, g, and b. For functions w and g, we apply Lemma 1 to obtain the full descriptions of $w_1, \ldots, w_k$ and $g_1, \ldots, g_{k-1}$ in $O(kn \log n)$ time. Then, we recursively compute the upper envelope of two upper envelopes: one for $w_1, \ldots, w_{\lfloor k/2 \rfloor}$ and the other for $w_{\lfloor k/2 \rfloor + 1}, \ldots, w_k$. This can be done in time linear to the complexity of the two upper envelopes, which is $O(kn)$ time, because any two sinusoidal curves cross $O(1)$ times in any subdomain of $(0, \pi)$. Since the recursion depth is bounded by $O(\log k)$, we can compute the full description of w in $O(kn \log k)$ time. Computing g can be done analogously. Note that functions w and g are piecewise sinusoidal with $O(kn)$ breakpoints, and function b is piecewise sinusoidal with $O(n)$ breakpoints.

Next, we specify the intervals of ρ-valid orientations. We call an orientation $\theta \in (0, \pi)$ *ρ-valid* if the gap-ratio $\rho(\theta)$ of $\mathsf{S}(\theta)$ is at least ρ. We can specify the intervals of ρ-valid orientations by solving equation $g(\theta) = \rho \cdot b(\theta)$. As both functions g and b are piecewise sinusoidal, we can find all the zeros of the equation in $O(kn)$ time, and these zeros are endpoints of the ρ-valid intervals. Note that the number of ρ-valid intervals is also bounded by $O(kn)$ since any two sinusoidal curves cross $O(1)$ times in any subdomain of $(0, \pi)$. At this stage, if there is no ρ-valid orientation, we report that there is no (k, ρ)-slab cover of P respecting R.

Finally, for each ρ-valid interval I, we minimize the width $w(\theta)$ of $\mathsf{S}(\theta)$ over $\theta \in I$. Since function w is piecewise sinusoidal with $O(kn)$ breakpoints and there are $O(kn)$ such intervals, we can find an optimal orientation θ^* that minimizes w over all ρ-valid orientations in total $O(kn)$ time.

Hence, the total running time is bounded by $O(kn \log n)$. It is not difficult to see that the space spent during the execution of the algorithm is $O(kn)$. $\square$

3.2 Reducing the space usage

In the following, we show how to reduce the space complexity of Lemma 2 to $O(n)$. This improved algorithm runs in an angular sweeping fashion by handling events and updating necessary invariants related to $\mathsf{S}(\theta)$ as θ increases from 0 to π.

Data structures, variables, and invariants. At any moment $\theta \in [0, \pi)$ during the execution of our algorithm, we maintain the following:

- A fully dynamic structure CH_i for each $i = 1, \ldots, k$. CH_i stores the convex hull of $P_i(\theta)$ and supports an extreme point query in a given direction, a tangent line query through a given point, a neighbor query, and an insertion/deletion in amortized $O(\log n)$ time using $O(n)$ space. Such a data structure is known by Brodal and Jacob [6].

- $2k$ lists $\mathsf{W}_1, \ldots, \mathsf{W}_k$, $\mathsf{G}_1, \ldots, \mathsf{G}_{k-1}$, and B. The list W_i stores sinusoidal functions with domain such that its tail stores the current sinusoidal form of function w_i and its predecessors store some previous sinusoidal pieces of w_i in the order. Similarly, G_i stores sinusoidal pieces of g_i, and B stores those of b. These $2k$ lists will be maintained so that the total number of elements does not exceed $5n$.

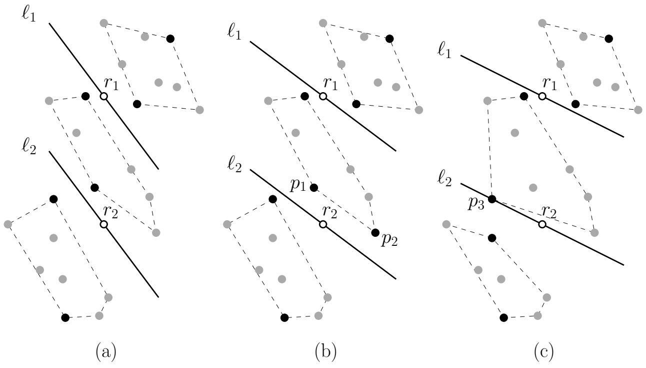


Figure 2: The convex hulls of $P_1(\theta)$, $P_2(\theta)$, and $P_3(\theta)$, respectively, are depicted by dashed lines, and the extreme points for each are shown as black dots. (b) A slab event occurs when a side p_1p_2 of the convex hull of $P_2(\theta)$ is parallel to the rotating lines. (c) A cross event occurs when ℓ_2 touches a point p_3.

- The two extreme points $q_i^+ = q_i^+(\theta)$ and $q_i^- = q_i^-(\theta)$ of $P_i(\theta)$ for each $i = 1, \dots, k$. If $P_i(\theta) = \emptyset$, these two variables are set to `nil`.

Events. Our *events* correspond to changes of the extreme points q_i^+ and q_i^- for $i = 1, \dots, k$. We distinguish two types of events:

- A *slab event* occurs when two or more points of P are contained in a boundary line of slab $\sigma_i(\theta)$ of $\mathsf{S}(\theta)$ for some i. Each slab event is associated with a tuple (ϕ, p^+, p^-, i), where ϕ is the orientation when the event occurs and (p^+, p^-) is the new pair of extreme points of $P_i(\theta)$ right after $\theta = \phi$.
- A *cross event* occurs when a point in P lies on $\ell_i(\theta)$ for some i. Each cross event is associated with a tuple (ϕ, p, i), where ϕ is the orientation at which the event occurs and the line $\ell_i(\theta)$ hits p at $\theta = \phi$.

Initialization. For the initialization, our algorithm sets $\theta = 0$ and computes all the above data structures and variables accordingly. Here, we extend the function $d_\theta(p, q)$ in such a way that $d_\theta(p, q) = 0$ when either p or q is `nil`. Recall that the points $r_1, \dots, r_{k-1}$ in the given separator R lie on a vertical line, so all the lines $\ell_1(0), \dots, \ell_{k-1}(0)$ coincide with the same vertical line. Hence, $P_1(0)$ consists of those in P lying on the right side and $P_k(0)$ the rest of those in P, while $P_2(0) = \cdots = P_{k-1}(0) = \emptyset$. For each $i = 1, \dots, k$, we insert the points in $P_i(0)$ into CH_i after initializing CH_i to be empty, and specify the pair of extreme points (q_i^+, q_i^-) of $P_i(0)$. We initialize the lists $\mathsf{W}_1, \dots, \mathsf{W}_k$, $\mathsf{G}_1, \dots, \mathsf{G}_{k-1}$, and B as empty lists. For each $i = 1, \dots, k$, we append a node with $(d_\theta(q_i^+, q_i^-), [0, \pi))$ to W_i. For each $i = 1, \dots, k-1$, and append a node with $(d_\theta(q_i^-, q_{i+1}^+), [0, \pi))$ to G_i. And, append a node with $(d_\theta(q_1^+, q_k^-), [0, \pi))$ to B.

Additionally, we need a priority queue Q, called the *event queue*, which will store at most $O(k)$ events which will occur, prioritized by their occurring orientations. Since only $P_1(0)$ and $P_k(0)$ can be non-empty, we create next slab events and cross events for $P_1(0)$ and $P_k(0)$ by tangent line queries and neighbor queries to CH_1 and CH_k, and insert them into Q.

Main loop. In the main loop of our algorithm, until the event queue Q becomes empty, we extract the next event from Q after the current orientation θ, handle the event, and evaluate partial width functions to compute an optimal solution. The last evaluation task is performed by procedure EVALUATE, which will be described later.

Let e be the event extracted from Q, and ϕ be the orientation of e. We handle e according to its type:

- Suppose e is a slab event associated with (ϕ, p^+, p^-, i). Then the pair of extreme points (q_i^+, q_i^-) of $P_i(\theta)$ has to change to (p^+, p^-) after ϕ. So we set q_i^+ to p^+ and q_i^- to p^-.

  Next, we update W_i. Let $(f, [\phi_0, \pi))$ be the tail of W_i. We modify it to $(f, [\phi_0, \phi))$ and append a new node $(d_\theta(p^+, p^-), [\phi, \pi))$ to the tail of W_i. Similarly, we update G_{i-1} and G_i if $1 < i < k$: Modify the right endpoint of the domain of the function at the tail of G_{i-1} (and also of G_i) to ϕ, and append a new node $(d_\theta(q_{i-1}^-, p^+), [\phi, \pi))$ to G_{i-1} and another $(d_\theta(p^-, q_{i+1}^+), [\phi, \pi))$ to G_i. If either $i = 1$ or $i = k$, we also modify B. If $i = 1$, we update G_1 as done above, modify the right endpoint of the domain of the function at the tail of B to ϕ, and append a new node $(d_\theta(p^+, q_k^-), [\phi, \pi))$ to B. If $i = k$, we update G_k as done above, modify the right endpoint of the domain of the function at the tail of B to ϕ, and append a new node $(d_\theta(q_1^+, p^-), [\phi, \pi))$ to B.

  Finally, we create the next possible slab event e' for $P_i(\theta)$ by finding the next extreme pair (q^+, q^-) of $P_i(\theta)$ as θ increases from ϕ. This can be done by two neighbor queries to the convex hull CH_i. To verify that this pair (q^+, q^-) indeed make the corresponding slab event occur at $\theta = \phi'$, we check the orientations of the tangents from r_i and r_{i-1} to CH_i by tangent line queries. Only if ϕ' is not bigger than these orientations of the tangents, the slab event for (q^+, q^-) occurs at ϕ'. In this case, we insert the slab event at ϕ' associated with (q^+, q^-, i).
- Suppose e is a cross event associated with (ϕ, p, i). In this case, p is about to cross line $\ell_i(\phi)$. We first update CH_i and CH_{i+1} by an insertion and a deletion of p. We then update q_i^- and q_{i+1}^+ correctly by extreme point queries to CH_i and CH_{i+1}.

Next, we update the function lists W_i, W_{i+1}, and G_i. For each of these list, if $(f, [\phi_0, \pi))$ is its tail, we modify it to $(f, [\phi_0, \phi))$. Also, we append $(d_\theta(q_i^+, q_i^-), [\phi, \pi))$ to W_i, $(d_\theta(q_{i+1}^+, q_{i+1}^-), [\phi, \pi))$ to W_{i+1}, and $(d_\theta(q_i^-, q_{i+1}^+), [\phi, \pi))$ to G_i.

Finally, we create the next cross event for $\ell_i(\theta)$ by tangent line queries to CH_i and CH_{i+1} and insert them into Q. Also, we create next slab events for $P_i(\theta)$ and for $P_{i+1}(\theta)$ with the updated extreme points, and insert each into Q after verification as described in handling a slab event.

After handling event e as above, we set θ to ϕ. If the event queue Q becomes empty, we set θ to π. Finally in the main loop, we call procedure EVALUATE only if the number of handled events is divisible by n or the event queue Q becomes empty.

Procedure EVALUATE. For each $\mathsf{L} \in \{\mathsf{W}_1, \ldots, \mathsf{W}_k, \mathsf{G}_1, \ldots, \mathsf{G}_{k-1}, \mathsf{B}\}$, let $(f, [\phi_0, \pi))$ be the tail of L. If $\phi_0 < \theta$, we modify the tail to $(f, [\phi_0, \theta))$ and append a new node $(f, [\theta, \pi))$ to the tail of L.

Observe that by construction the left endpoint of the domain of the function stored at the head of L is equal to some $\theta_0 < \theta$ for every list L, and the union of the domains of all nodes except the tail is exactly $[\theta_0, \theta)$. Therefore, for each $i = 1, \ldots, k$, the partial functions stored at the nodes of W_i, except its tail, form the function w_i restricted to domain $[\theta_0, \theta)$; for each $i = 1, \ldots, k-1$, those of G_i, except its tail, form the function g_i restricted to $[\theta_0, \theta)$; those of B, except its tail, form the function b restricted to $[\theta_0, \theta)$.

We compute the upper envelope w of w_i's restricted in domain $[\theta_0, \theta)$ and also the lower envelope g of g_i's restricted in domain $[\theta_0, \theta)$ as done in Lemma 2. Then, we consider the ρ-valid intervals in $[\theta_0, \theta)$ and maximize w in each ρ-valid interval. This way, we compute the minimum possible width of (k, ρ)-slab covers of P in the sub-domain $[\theta_0, \theta)$.

Finally, we delete all the nodes except the tail from each of the lists $\mathsf{W}_1, \ldots, \mathsf{W}_k$, $\mathsf{G}_1, \ldots, \mathsf{G}_{k-1}, \mathsf{B}$.

Analysis. The initialization step can be done in $O(n \log n)$ time. The main loop, except the call of procedure EVALUATE takes $O(\log n)$ time per event. The number of handled events is bounded by $O(kn)$ by Lemma 1 since each event corresponds to one breakpoint of at most three of the functions $w_1, \ldots, w_k, g_1, \ldots, g_{k-1}, b$. Thus, the total time for handling events is bounded by $O(kn \log n)$. Since procedure EVALUATE is called every n events and at most three nodes are appended to the lists of functions for each event, the total number of nodes stored in the lists does not exceed $2k + 3n \le 5n$ at each call of EVALUATE. Hence, the time spent for a call of EVALUATE is bounded by $O(n \log k)$ and the total time spend for EVALUATE is $O(kn \log k)$ since the number of calls is $O(k)$.

The structures $\mathsf{CH}_1, \ldots, \mathsf{CH}_k$ use $O(n)$ space in total [6]. The number of events stored in Q is bounded by $O(k)$ at any time. The total number of nodes stored in the function lists $\mathsf{W}_1, \ldots, \mathsf{W}_k$, $\mathsf{G}_1, \ldots, \mathsf{G}_{k-1}$, and B is bounded by $5n$ as analyzed above. Hence, the total space complexity is bounded by $O(n)$.

We finally conclude the following theorem.

Theorem 3 *Given a set P of n points, an integer $k \ge 2$, a real number $0 < \rho \le 1$, and a separator R, a minimum-width (k, ρ)-slab cover of P respecting R can be computed in $O(kn \log n)$ time and $O(n)$ space, if exists. Otherwise, it is reported in the same time bound that there is no (k, ρ)-slab cover of P that respects R.*

4 Computing a Minimum-Width (k, ρ)-Slab Cover

In this section, we show how to compute a minimum-width (k, ρ)-slab cover of P in Problem 1. Recall that a real ρ is given, and it holds that $0 < \rho \le 1/(k-1)$; otherwise, we just report that there is no (k, ρ)-slab cover of P.

The *directional width* of P in orientation $\theta \in [0, \pi)$ is the width of the smallest slab enclosing P in orientation θ, denoted by $d_\theta(P) := \max_{p,q \in P} d_\theta(p, q)$. A subset $K \subseteq P$ is called an *ϵ-coreset* for the directional width of P if $d_\theta(K) \ge (1 - \epsilon) d_\theta(P)$ for any orientation θ.

Our algorithm starts by computing a $(\rho/2)$-coreset $K \subseteq P$ for the directional width of P. Then, for each antipodal pair (p, q) of the convex hull of K, we generate candidate separators from the segment pq and apply the algorithm described in Theorem 3. In the following, we mainly focus on the correctness of our algorithm by proving that any (k, ρ)-slab cover of P indeed respects one of our generated separators.

4.1 Candidate separators from a coreset

Let $K \subseteq P$ be a $(\rho/2)$-coreset $K \subseteq P$ for the directional width of P.

Lemma 4 *For any (k, ρ)-slab cover $\mathsf{S} = (\sigma_1, \ldots, \sigma_k)$ of P, it holds that $K \cap \sigma_1 \neq \emptyset$ and $K \cap \sigma_k \neq \emptyset$.*

Proof. Suppose for a contradiction that there exists a (k, ρ)-slab cover $\mathsf{S}' = (\sigma'_1, \ldots, \sigma'_k)$ with gaps $\gamma'_1, \ldots, \gamma'_{k-1}$ such that $K \cap \sigma'_1 = \emptyset$ or $K \cap \sigma'_k = \emptyset$. Without loss of generality, we assume that $K \cap \sigma'_1 = \emptyset$. Let θ' denote the orientation of S'. We then have

$$d_{\theta'}(K) \ge (1 - \rho/2) b(\mathsf{S}')$$

on one hand, since K is a $(\rho/2)$-coreset of P and $b(\mathsf{S}') = d_{\theta'}(P)$. On the other hand, the assumption that $K \cap \sigma'_1 = \emptyset$ implies that

$$d_{\theta'}(K) \le b(\mathsf{S}') - (w(\sigma'_1) + w(\gamma'_1)).$$

Plugging these two inequalities, we have

$$g(\mathsf{S}') \le w(\gamma_1') \le \rho/2 \cdot b(\mathsf{S}') - w(\sigma_1') < \rho \cdot b(\mathsf{S}'),$$

and thus $\rho(\mathsf{S}') = g(\mathsf{S}')/b(\mathsf{S}') < \rho$, a contradiction. □

Furthermore, in Lemma 4, we can pick two points $p \in K \cap \sigma_1$ and $q \in K \cap \sigma_k$ such that (p, q) forms an antipodal pair of the convex hull of K.

Corollary 5 *For any (k, ρ)-slab cover $\mathsf{S} = (\sigma_1, \ldots, \sigma_k)$ of P, there is an antipodal pair (p, q) of K such that $p \in \sigma_1$ and $q \in \sigma_k$.*

For any two points p, q in the plane, let R_{pq} be the set of $\lceil 1/\rho \rceil$ points $p_j \in pq$ such that the distance from p to p_j is exactly $j \cdot |pq|/(\lceil 1/\rho \rceil + 1)$ for $1 \le j \le \lceil 1/\rho \rceil$.

Lemma 6 *For any (k, ρ)-slab cover $\mathsf{S} = (\sigma_1, \ldots, \sigma_k)$ of P, let p and q be any two points such that $p \in \sigma_1$ and $q \in \sigma_k$. Then, each gap of S contains at least one point in R_{pq}. Therefore, there is a separator $\mathsf{R} = (r_1, \ldots, r_{k-1})$ of S such that $r_i \in R_{pq}$ for all $1 \le i \le k - 1$.*

Proof. Let θ be the orientation of S and $\gamma_1, \ldots, \gamma_{k-1}$ be the gaps of S. Suppose for a contradiction that $\gamma_i \cap R_{pq} = \emptyset$ for some i with $1 \le i \le k - 1$. Then, its width $w(\gamma_i)$ is small in the sense that

$$w(\gamma_i) \le \frac{d_\theta(p, q)}{\lceil 1/\rho \rceil + 1} < \rho \cdot d_\theta(p, q).$$

Since p and q are contained in the k-slab cover S, we also have $d_\theta(p, q) \le b(\mathsf{S})$, which implies that

$$g(\mathsf{S}) \le w(\gamma_i) < \rho \cdot b(\mathsf{S}),$$

a contradiction to the assumption that $\rho(\mathsf{S}) \ge \rho$. □

4.2 Algorithm

Now, we are ready to describe our algorithm. Our algorithm first computes a $(\rho/2)$-coreset $K \subseteq P$ of size $O(1/\rho)$ for the directional width of P in $O(n)$ time [10, Chapter 20]. Then, it computes all antipodal pairs of the convex hull of K by using the rotating caliper method [14] after computing the convex hull of K. Finally, for each of these antipodal pair (p, q), it generates all possible $(k - 1)$-combinations from the set R_{pq}, as candidate separators, and computes a minimum-width (k, ρ)-slab cover of P by applying Theorem 3.

The correctness of our algorithm is guaranteed by the above discussion through Lemmas 4 and 6. Since K is a subset of P and $|K| = O(1/\rho)$, the number of antipodal pairs is also bounded by $O(1/\rho)$. Hence, the number of generated candidate separators is bounded by $O(1/\rho^k)$. The following theorem summarizes the result.

Theorem 7 *Given a set P of n points in the plane, an integer $k \ge 2$, and a real $0 < \rho \le 1$, a minimum-width (k, ρ)-slab cover of P can be computed in $O(\rho^{-k} \cdot kn \log n)$ time and $O(n)$ space, if exists. Otherwise, it is reported in the same time bound that there is no (k, ρ)-slab cover of P.*

5 Computing a Maximum-Gap-Ratio k-Slab Cover

In this section, we present an algorithm computing a maximum-gap-ratio k-slab cover of P in Problem 2. Let $\rho_{\max}$ be the maximum possible real number such that there exists a $(k, \rho_{\max})$-slab cover of P.

Observe that procedure Evaluate in the algorithm of Theorem 3 can be easily modified for maximizing the gap-ratio, instead of minimizing the width. This is possible because procedure Evaluate indeed specifies all necessary functions $g(\theta)$ and $b(\theta)$ explicitly. With this modified version of procedure Evaluate, we have the following corollary of Theorem 7.

Corollary 8 *Given a set P of n points in the plane, an integer $k \ge 2$, and a real $0 < \rho \le 1$, a k-slab cover of P with maximum possible gap-ratio $\rho_{\max}$ can be computed in $O(\rho^{-k} kn \log n)$ time and $O(n)$ space, if $\rho \le \rho_{\max}$. Otherwise, it is reported in the same time bound that there is no (k, ρ)-slab cover of P.*

Thus, what remains is to find a value ρ with $0 < \rho < \rho_{\max}$, which is big enough. For any integer $i \ge 0$, let $\rho_i = 2^{-i/k}$. We then search for the integer t such that $\rho_{t-1} > \rho_{\max} \ge \rho_t$. This can be done by running the algorithm in Corollary 8 with $\rho = \rho_i$ for $i = 0, 1, \ldots$ in this order until it first outputs a k-slab cover of P. Then, the resulting k-slab cover of P indeed the one with maximum possible gap-ratio by Corollary 8.

In order to analyze the running time, observe that the time spent by the t applications of Corollary 8 is bounded by

$$\begin{aligned}
&\sum_{i=0,\ldots,t} O(\rho_i^{-k} \cdot kn \log n) \\
&= \sum_{i=0,\ldots,t} O(2^i \cdot kn \log n) \\
&= O(2^t \cdot kn \log n) \\
&= O(\rho_t^{-k} \cdot kn \log n),
\end{aligned}$$

since $\rho_i^{-k} = 2^i$. In addition, we have $\rho_t^{-k} < 2\rho_{\max}^{-k}$ since $\rho_{t-1} > \rho_{\max} \ge \rho_t$. This implies that the time complexity is bounded by $O(\rho_{\max}^{-k} \cdot kn \log n)$.

Theorem 9 *Given a set P of n points and an integer $k \ge 2$, a maximum-gap-ratio k-slab cover of P can be computed in $O(\rho_{\max}^{-k} \cdot kn \log n)$ time and $O(n)$ space, where $\rho_{\max}$ is the maximum possible gap-ratio of a k-slab cover of P.*

References

[1] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. A $(1+\epsilon)$-approximation algorithm for 2-line-center. *Computational Geometry*, 26(2):119–128, 2003.

[2] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for a k-line center. *Algorithmica*, 42:221–230, 2005.

[3] P. K. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11:185–195, 1994.

[4] P. Alevizos, J.-D. Boissonnat, and F. Preparata. An optimal algorithm for the boundary of a cell in a union of rays. *Algorithmica*, 5:573–590, 1990.

[5] S. W. Bae. Minimum-width double-strip and parallelogram annulus. *Theoretical Computer Science*, 833:133–146, 2020.

[6] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Symposium on Foundations of Computer Science*, pages 617–626. IEEE Computer Society, 2002.

[7] B. Chazelle, L. Guibas, and D.-T. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25(1):76–90, 1989.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Alogorithms and Applications.* Springer-Verlag, 2nd edition, 2000.

[9] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[10] S. Har-Peled. *Geometric Approximation Algorithms.* American Mathematical Society, USA, 2011.

[11] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):761–765, 1988.

[12] J. W. Jaromczyk and M. Kowaluk. The two-line center problem from a polar view: A new algorithm and data structure. In *Workshop on Algorithms and Data Structures*, pages 13–25. Springer, 1995.

[13] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations Research Letters*, 1:194–197, 1982.

[14] G. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON*, 1983.

[15] H. Wang. A simple algorithm for computing the zone of a line in an arrangement of lines. In *SIAM Symposium on Simplicity in Algorithms*, pages 79–86. SIAM, 2022.

Improved Algorithms for Burning Planar Point Sets[*]

Shahin Kamali[†] Mohammadmasoud Shabanijou[‡]

Abstract

Given a set $\mathcal{P}$ of points in the plane, the geometric burning problem asks for minimizing the number of rounds to "burn" $\mathcal{P}$. It is possible to burn $\mathcal{P}$ in k rounds if one can cover all points in $\mathcal{P}$ with k disks of distinct radii from $\{0, 1, \ldots, k-1\}$. In the *anywhere burning* variant of the problem, the disks can be located at any position in the plane, while in the *point burning* variant, they must be centred at points of $\mathcal{P}$. Both variants are NP-hard, and the best existing algorithms have approximation factors of respectively $1.92188+\epsilon$ and $1.96296+\epsilon$ for anywhere and point burning [Gokhale et al. WALCOM 2023]. In this paper, we present techniques for improving these algorithms. We first present a simple anywhere burning algorithm with an approximation factor of at most $1.865 + \epsilon$, and then improve this algorithm to achieve an approximation factor of $11/6 + \epsilon \approx 1.833 + \epsilon$ for arbitrary small $\epsilon > 0$. We also present a point-burning algorithm with an improved approximation factor of at most $1.944 + \epsilon$.

1 Introduction

Graph burning is a simple model for the spread of social influence in networks [3]. Given an undirected, unweighted graph G, the objective is to measure how quickly "fire", representing gossip or a piece of fake news, can be spread in G. The burning process occurs in discrete rounds. All vertices of G are initially unburned. At each round, a new fire starts at a selected vertex of G, called a *source*, while the existing fires extend to their adjacent vertices. The burning graph problem asks for sources that minimize the number of rounds to burn all vertices of G, also known as the *burning number* of G.

Kiel et al. [9] introduced a geometric variant of the graph burning problem. The input is a set $\mathcal{P}$ of points in the plane, and the goal is to burn all these points in a minimum number of rounds. Initially, all points are unburned. At each round, a new fire breaks out at a new point in the plane, a source, while the existing fires extend to all points within a unit distance from the currently burned points. As before, the goal is to minimize the number of rounds to burn all points in $\mathcal{P}$. In the *anywhere burning* variant of the problem, the sources can be *any* point in the plane, while in the point burning, the sources must belong to $\mathcal{P}$.

Figure 1 illustrates anywhere and point burning models. Note that if a burning process completes in k rounds, the fire started at the beginning of round i, for any $i \in \{1, 2, \ldots, k\}$, burns vertices within distance $k-i$ of the source selected at round i. Therefore, the burning problem is equivalent to selecting a minimum number of disks with distinct radii in the set $\{0, 1, \ldots, k-1\}$ that cover all nodes in the input set $\mathcal{P}$. In anywhere burning, these disks can be located at any position in the plane, while in point burning, they must be centred at vertices of $\mathcal{P}$.

1.1 Previous Work and Contribution

The graph burning problem is NP-hard, even for simple graph families like trees and disjoint path forests [1]. The best existing algorithm for general graphs has an approximation factor of 3 [1, 4], which can be slightly improved to $3-1/b(G)$ [6], where $b(G)$ denotes the burning number of the input graph. There are polynomial-time approximation schemes (PTASs) for disjoint path forests [4], trees, and, more generally, graphs of bounded treewidth [11]. Improved algorithms exist for other graph families, e.g., interval graphs [8]. We refer to [2] for a survey of the results on the graph burning problem.

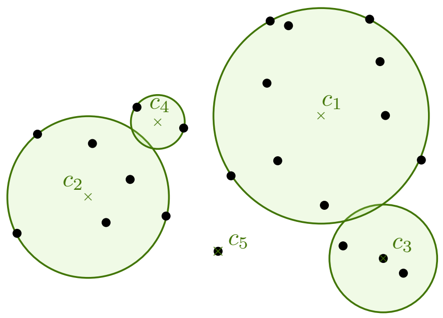
In the geometric setting, both anywhere pointing and point burning are NP-hard [9]. Kiel et al. leveraged an existing polynomial-time approximation scheme (PTAS) for the *discrete unit disk cover* (DUDC) problem [12] to achieve algorithms with approximation factors of $2+\epsilon$ for both anywhere burning and point burning problems. These approximation factors were later improved by Gokhale et al. [7] to $1.92188 + \epsilon$ for anywhere burning and $1.96296 + \epsilon$ for point burning. In the 1-dimensional setting, when all points are colinear, both problem variants admit a PTAS [7].

This paper presents new algorithms with improved approximation factors for anywhere and point burning. For anywhere burning, we relate the problem to the *disk covering* (DC) problem [10], which asks for covering a disk of radius $r > 1$ with a minimum number of unit disks. Leveraging the existing result for the disk covering problem, we introduce a simple anywhere burn-

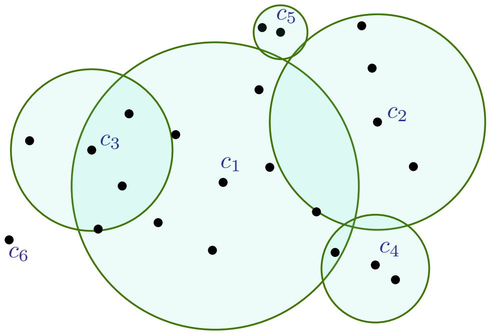
[*]This work is funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[†]Department of Electrical Engineering and Computer Science, York University, kamalis@yorku.ca

[‡]Department of Computer Science, University of Manitoba, shabanim@myumanitoba.ca



(a) An anywhere burning solution



(b) A point burning solution

Figure 1: An illustration of the burning protocols. Here, c_i indicate the i'th source. In anywhere pointing, the sources can be located at any point in the plane, while in point burning, they must belong to the input point set. In this example, anywhere burning and point burning take 5 and 6 rounds, respectively.

ing algorithm with an approximation factor of at most $1.865+\epsilon$ (Theorem 1) for any arbitrary small $\epsilon > 0$. We improve this approximation factor to $11/6 + \epsilon$ with a slightly more complicated algorithm that is based on a "mix-and-match" approach for covering a disk of radius r with smaller disks of various radii $< r$ (Theorem 2). For the point burning problem, we build on the ideas of [7] to present an algorithm with an improved approximation factor of $1.944 + \epsilon$ (Theorem 3). Throughout the paper, we use $[a, b)$ to denote the set of integers like x such that $a \leq x < b$.

1.2 Discrete Unit Disk Cover (DUDC) Problem

An instance $(\mathcal{P}, (C, r))$ of the discrete unit disk cover (DUDC) problem is defined by a set $\mathcal{P}$ of points in the plane and a set (C, r) of disks of uniform radii r centred at the point set C. The objective is to cover all points in $\mathcal{P}$ with a minimum number of disks from (C, r). Mustafa and Ray provided a PTAS for the DUDC problem [12]. All our approximation algorithms for planar burning use this result as a black box.

2 Anywhere Burning Problem

Before describing our algorithm for anywhere burning, we review the framework of Kiel et al. [9], which we will use in our algorithms. Given a set $\mathcal{P}$ of n points as an instance of the anywhere burning problem, Kiel et al. [9] show that there exists an optimal solution where each burning source either coincides with a given point in $\mathcal{P}$ or lies at the center of a circle determined by two or three given points of $\mathcal{P}$. Consider a point set C formed as the union of $\mathcal{P}$ and the centers of all these circles. Therefore, C has a polynomial number of points, namely at most $n+\binom{n}{2}+\binom{n}{3}$ points. One can define instances $(\mathcal{P}, (C, g))$ of the DUDC problem that asks for covering points in $\mathcal{P}$ with a minimum subset of disks in (C, g), that is, disks centred at points of C and of uniform radius g. Here, g is a *guess* that takes integer values in $[1, n]$. For a fixed value g, one can use the result of [12] to solve the instance $(\mathcal{P}, (C, g))$ of the DUDC problem. There are two possibilities to consider:

(1) Suppose the number of disks in the PTAS output is larger than g. We can conclude that at least $\lfloor g/(1 + \epsilon)\rfloor$ disks from the set (C, g) of disks are necessary to cover all points in $\mathcal{P}$. In other words, no subset of size $\lfloor g/(1+\epsilon)\rfloor-1$ from disks of (C, g) can cover all points in $\mathcal{P}$. Therefore, no subset of $\lfloor g/(1 + \epsilon)\rfloor - 1$ disks of smaller distinct radii from $\{0, 1, \dots, \lfloor g/(1+\epsilon)\rfloor-2\}$ can cover all points of $\mathcal{P}$. We may conclude that burning $\mathcal{P}$ is not possible in $\lfloor g/(1 + \epsilon)\rfloor - 1$ rounds and thus takes at least $\lfloor g/(1 + \epsilon)\rfloor$ rounds.

(2) Suppose the number of disks in the PTAS output is at most g. That is, we can cover points of $\mathcal{P}$ with g disks (of radius g) from (C, g). Consequently, it is possible to cover points of $\mathcal{P}$ with $2g$ disks of distinct radii $\{0, 1, \dots, 2g - 1\}$ by only using disks of radius $\geq g$. We can conclude that it is possible to burn all points in $\mathcal{P}$ in $2g$ rounds.

Let g^* denote the smallest guess value such that the PTAS output for $(\mathcal{P}, (C, g^*))$ is of size at most g^*. From the above observations, at least $\lfloor(g^*-1)/(1+\epsilon)\rfloor$ rounds are necessary to burn the instance $\mathcal{P}$ of the burning problem; this is because the PTAS output for guess $g^* - 1$ has been of size larger than $g^* - 1$. On the other hand, $2g^*$ rounds are sufficient to burn $\mathcal{P}$. The approximation factor of the resulting solution is thus $2g^*/\lfloor(g^* - 1)/(1 + \epsilon)\rfloor$, which approaches $2(1 + \epsilon)$ for large values of g^*. Note that if g^* is a constant, then the burning number is also a constant. Therefore, one can find an optimal solution in polynomial time via an exhaustive approach by trying all subsets of C and all possible placements of disks of radii $\{0, 1, \dots, g\}$ on the selected subset. The smallest value of g that covers all points of $\mathcal{P}$ gives the optimal burning solution.

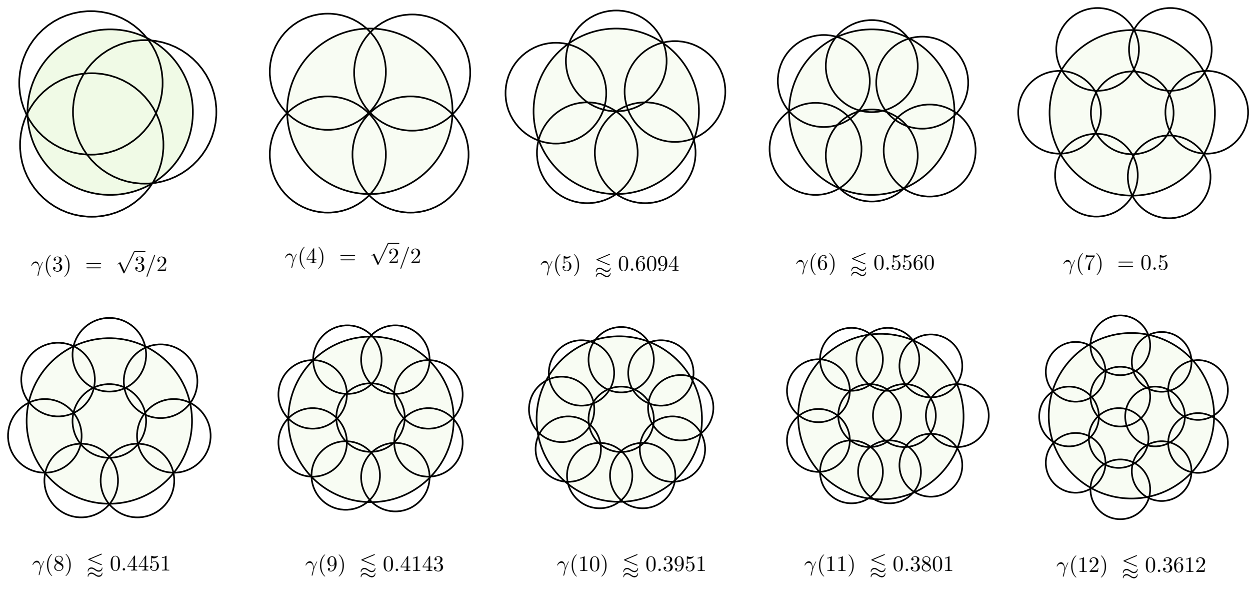

$\gamma(3) = \sqrt{3}/2$
$\gamma(4) = \sqrt{2}/2$
$\gamma(5) \lessapprox 0.6094$
$\gamma(6) \lessapprox 0.5560$
$\gamma(7) = 0.5$
$\gamma(8) \lessapprox 0.4451$
$\gamma(9) \lessapprox 0.4143$
$\gamma(10) \lessapprox 0.3951$
$\gamma(11) \lessapprox 0.3801$
$\gamma(12) \lessapprox 0.3612$

Figure 2: [5] A summary of the optimal solutions for the disk covering (DC) problem for $k \in [3, 12]$

The above argument was used by Kiel et al. [9] to prove an approximation factor of $2+\epsilon$ for the anywhere burning problem. In our algorithms, we improve over Step 2 of this algorithm to achieve better approximation factors. This is achieved by using disks of radii smaller than g^* to cover some of the disks in the output by the PTAS for the DUDC instances.

2.1 Disk-Cover-Burn (DCB) Algorithm

In this section, we present an algorithm with an approximation factor of $1.865+\epsilon$, which is inspired by the disk covering (DC) problem [10]. The DC problem asks for the smallest real number $\gamma(k)$ such that k disks of radius $\gamma(k)$ can be arranged in such a way as to cover the unit disk. For example, we have $\gamma(3) = \sqrt{3}/2$ [5], which indicates that it is possible to cover a unit disk with three disks of radius $\sqrt{3}/2$ (and it is not possible with smaller disks). The Disk Covering problem has a long history (see, e.g., [13, 10, 16, 15] for some results and [14] for a summary). Figure 2 shows the best coverings for $k \in [3, 12]$.

We use the existing results for the disk covering problem to improve the algorithm of [9] for burning a set $\mathcal{P}$ of points. Let $\epsilon > 0$ be an arbitrarily small value, and define $\epsilon' = \epsilon/1.865$. Given an input $\mathcal{P}$ of points in the plane, form a point set C of $n + \binom{n}{2} + \binom{n}{3}$ points as described in Section 2. We repeatedly apply the PTAS for the DUDC problem with parameter ϵ' to find the smallest value g^* for which the PTAS on instance $(\mathcal{P}, (C, g^*))$ returns a set U of at most g^* disks (of uniform radius g^*). If g^* is a constant, we apply an exhaustive approach to find an optimal burning scheme in polynomial time. In what follows, we assume g^* is arbitrarily large.

Let D be a set of $\lfloor 1.865g^* \rfloor$ disks with distinct radii $\{0, 1, \ldots, \lfloor 1.865g^* \rfloor - 1\}$. We explain how to cover the disks in U with disks in D. Recall that there are g^* disks of uniform radius g^* in U. We classify disks in D based on their radii. Disks with radii at least g^* belong to Class 0. There are $\lfloor 0.865g^* \rfloor$ disks of Class 0, and each can cover one disk from U. There is no disk of Class 1 or 2. Disks with radii in $\left[\lceil \sqrt{3}g^*/2 \rceil, g^*\right)$ belong to Class 3, and each group of 3 of them can cover a disk of U because these disks' radii are at least $\lceil \sqrt{3}g^*/2 \rceil \geq \gamma(3)g^*$. Given that there are $g^* - \lceil \sqrt{3}g^*/2 \rceil \geq \lfloor (2 - \sqrt{3})g^*/2 \rfloor$ such disks, they can cover $\lfloor (2-\sqrt{3})g^*/6 \rfloor$ disks from U. Similarly, for any $i \in [4, 12]$, disks of D with radii in $[\lceil \gamma(i)g^* \rceil, \lceil \gamma(i-1)g^* \rceil)$ belong to class i and each group of i of them can cover a disk of U. Given that there are $(\lceil \gamma(i-1) \rceil - \lceil \gamma(i) \rceil)g^*$ such disks, they can cover $\lfloor (\gamma(i-1) - \gamma(i))g^*/i \rfloor$ disks from U. The details for the number of disks in each class of D and the number of disks from U that they cover are provided in Table 1. The total number of disks that can be collectively covered by all classes in D is at least $1.0009g^* - 11$, which is enough to cover all g^* disks of U, assuming large g^*. We can conclude the following theorem.

Theorem 1 Disk-Cover-Burn *has an approximation factor of at most* $1.865+\epsilon$*, for an arbitrary small* $\epsilon > 0$.

Proof. Since g^* is the smallest value for which the PTAS for the DUDC instance $(\mathcal{P}, (C, g))$ with parameter $\epsilon' = \epsilon/1.865$ returns an output with at most g^* disks, the burning number of the input set $\mathcal{P}$ is at least $(g^*-1)/(1+\epsilon')$. From the outcome for the PTAS, we know that all points in $\mathcal{P}$ can be covered with the set

i	radius range	covered disks from U
0	$\geq g^*$	$\lfloor 0.865g^* \rfloor$
3	$[\sqrt{3}g^*/2, g^*)$	$\lfloor (2-\sqrt{3})g^*/6 \rfloor$
4	$[\sqrt{2}g^*/2, \sqrt{3}g^*/2)$	$\lfloor (\sqrt{3}-\sqrt{2})g^*/8 \rfloor$
5	$[0.6094g^*, \sqrt{2}g^*/2)$	$\lfloor (\sqrt{2}/2 - 0.6094)g^*/5 \rfloor$
6	$[0.5560g^*, 0.6094g^*)$	$\lfloor 0.0534g^*/6 \rfloor$
7	$[0.5g^*, 0.5560g^*)$	$\lfloor 0.0560g^*/7 \rfloor$
8	$[0.4451g^*, 0.5g^*)$	$\lfloor 0.0549g^*/8 \rfloor$
9	$[0.4143g^*, 0.4451g^*)$	$\lfloor 0.0308g^*/9 \rfloor$
10	$[0.3950, 0.4143)$	$\lfloor 0.0193g^*/10 \rfloor$
11	$[0.3801, 0.3950)$	$\lfloor 0.0149g^*/11 \rfloor$
12	$[0.3612, 0.3801)$	$\lfloor 0.0189g^*/12 \rfloor$
sum	–	$> 1.0009g^* - 11 > g^*$

Table 1: A summary of disk classification by Disk-Cover-Burn. The total number of covered disks from U (sum of the numbers in the last column) is larger than g^* and hence disks in D can be used to cover all disks of U.

U of g^* disks of radius g^*. As discussed above, Disk-Cover-Burn covers these disks using $\lfloor 1.865g^* \rfloor$ disks of radii $\{0, 1, \ldots, \lfloor 1.865g^* \rfloor - 1\}$, which gives a burning scheme that completes within $\lfloor 1.865g^* \rfloor$ rounds. Therefore, the approximation factor of Disk-Cover-Burn is at most $\frac{1.865g^*}{(g^*-1)/(1+\epsilon')}$, which converges to $1.865(1+\epsilon') = 1.865 + \epsilon$ for large values of g^*. □

2.2 Improved-Disk-Cover-Burn (IDCB) Algorithm

In this section, we modify Disk-Cover-Burn to improve its approximation factor from 1.865 to $11/6 \approx 1.833$. Note that Disk-Cover-Burn covers each disk in the set U given by the PTAS for the DUDC problem with disks that belong to the *same* class (disks with almost equal radii). Our new algorithm, Improved-Disk-Cover-Burn, uses a "mix-and-match" approach to cover disks of U. In other words, each disk in U is covered with disks of different radii, allowing for a more efficient covering of U.

Given a fixed value $\epsilon > 0$, Improved-Disk-Cover-Burn starts with finding the smallest guess value h^* for which the PTAS output for the DUDC problem with parameter $\epsilon' = 6\epsilon/11$ is formed by a set U of at most h^* disks of radius h^*. If h^* is a constant, we use an exhaustive approach to find the optimal solution for the burning problem in polynomial time. Thus, in what follows, we assume h^* is arbitrarily large and define g^* as the smallest integer divisible by 60 greater than or equal to h^* (thus, $g^* \leq h^* + 59$). We show that it is possible to cover the disks in U with a set D of disks of radii $\{0, 1, \ldots, 11g^*/6 - 1\}$.

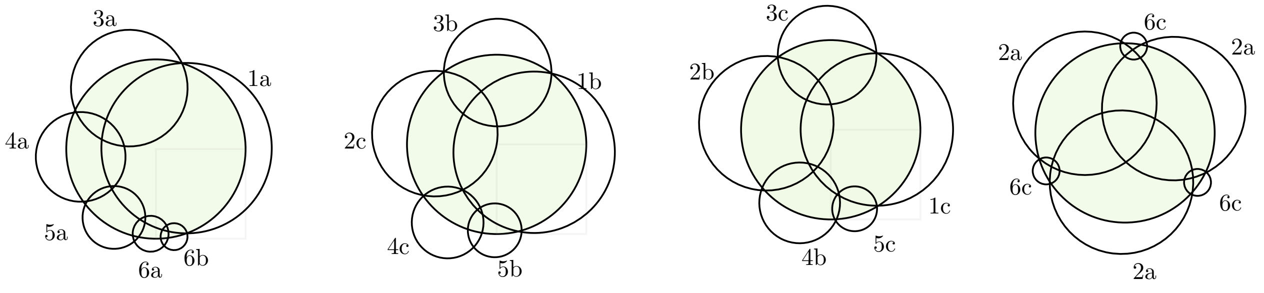
Like Disk-Cover-Burn, we classify disks in D based on their radii; but our classification differs. We partition D into eight classes. Disks of radius g^* or larger belong to Class 0. For $i \in \{1, 2, \ldots, 6\}$, disks of D with radius in $(g^*(1-0.15i), g^*(1.15-0.15i)]$ belong to class i. For example, disks with radius in $[0.85g^*, g^*)$ belong to Class 1, and disks with radius in $[0.7g^*, 0.85g^*)$ belong to Class 2. Disks of radius smaller than $0.1g^*$ belong to Class 7 and are not used in covering U. For $i \in [1, 6]$, we further partition disks of class i into subclasses (ia), formed by the largest one-third of the disks in class i, (ib), formed by the middle one-third, and (ic), formed by the smallest one-third of disks in class i. For example, disks of class (1a) have radius at least $0.95g^*$, disks of Class 1b have radius in $[0.9g^*, 0.95g^*)$ and disks of Class 1c have radius in $[0.85g^*, 0.9g^*)$. See Figure 3 for a summary of classes.

Based on this classification, we define *groups* of disks from D as follows. A group of type 0 is formed by one disk of Class 0 (with a radius of at least g^*; there are $5g^*/6$ such disks. A group of type 1 contains one disk from each of the classes $1a, 3a, 4a, 5a, 6a$, and $6b$. A group of type 2 has one disk from each of the classes $1b, 2c, 3b, 4c$, and $5b$. A group of type 3 contains one disk from each of the classes $1c, 2b, 3c, 4b$, and $5c$. Given that at most one item from each class appear in each group, and there are $0.05g^*$ disks of each subclass in D, we can form $0.05g^*$ disjoint groups of type t for $t \in \{1, 2, 3\}$. A group of type 4 is formed by three disks of class $2a$ and 3 disks of class $6c$; given that three members of each class $2a$ and $6c$ are present in one group, we can form $0.05g^*/3$ groups of type 4. In total, there are (i) $5g^*/6$ groups of type 0, (ii) $3 \cdot (0.05g^*)$ groups of type 1, 2, or 3, and (iii) $0.05g^*/3$ groups of type 4. Therefore, D is partitioned into $5g^*/6 + 0.15g^* + 0.05g^*/3 = g^*$ disjoint groups. Moreover, disks of each group can each cover one disk from U. Let S be any disk of U (of radius g^*). Groups of type 0 include a disk of radius at least g^*, which can cover S. For $i \in \{1, 2, 3, 4\}$, it suffices to position the disks of each group in a way that a chord of S passes through their centers. Figure 3 shows the positioning of the smallest disks in each group that ensures covering of U. We can conclude the following theorem:

Theorem 2 Improved-Disk-Cover-Burn *has an approximation factor of at most* $11/6 + \epsilon$ *for any arbitrary small* $\epsilon > 0$.

Proof. Recall that h^* is the smallest value for which the PTAS for the DUDC instance with parameter $\epsilon' = 6\epsilon/11$ has an output with at most h^* disks. Therefore, the burning number of the input set $\mathcal{P}$ is at least $(h^* - 1)/(1+\epsilon')$. Given that $h^* \geq g^* - 59$, burning $\mathcal{P}$ requires at least $(g^* - 60)/(1+\epsilon')$ rounds.

From the outcome for the PTAS, we know that all points in $\mathcal{P}$ can be covered with the set U of h^* disks of radius h^* and thus g^* disks of radius g^* (since $g^* \geq h^*$). As discussed above, Disk-Cover-Burn covers all disks



Group 1 covering Group 2 covering Group 3 covering Group 4 covering

Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
1a: $[0.95g^*, g^*]$	2a: $[0.8g^*, 0.85g^*)$	3a: $[0.65g^*, 0.7g^*)$	4a: $[0.5g^*, 0.55g^*)$	5a: $[0.35g^*, 0.4g^*)$	6a: $[0.2g^*, 0.25g^*)$
1b: $[0.9g^*, 0.95g^*]$	2b: $[0.75g^*, 0.8g^*)$	3b: $[0.6g^*, 0.65g^*)$	4b: $[0.45g^*, 0.5g^*)$	5b: $[0.3g^*, 0.35g^*)$	6b: $[0.15g^*, 0.2g^*)$
1c: $[0.85g^*, 0.9g^*]$	2c: $[0.7g^*, 0.75g^*)$	3c: $[0.55g^*, 0.6g^*)$	4c: $[0.4g^*, 0.45g^*)$	5c: $[0.25g^*, 0.3g^*)$	6c: $[0.1g^*, 0.15g^*)$

Figure 3: Disk classification by IMPROVED-DISK-COVER-BURN and covering bins of U using disks of each group. Each highlighted disk has radius g^*, and the radii of disks that cover them are the smallest radius from their respective class.

in a set U formed by g^* disks of radius g^* with $11g^*/6$ disks of radii $\{0, 1, \dots \lfloor 11g^*/6 \rfloor - 1\}$. This ensures a burning scheme that completes within $11g^*/6$ rounds. The approximation factor of IMPROVED-DISK-COVER-BURN is thus at most $\frac{\lfloor 11g^*/6 \rfloor}{(g^*-60)/(1+\epsilon')}$, which converges to $11/6(1 + \epsilon') = 11/6 + \epsilon$ for large values of g^*. □

3 Point Burning Problem

In this section, we present an algorithm for pointing burning with an approximation factor of at most $1.944 + \epsilon$, which is an improvement over $1.96296 + \epsilon$ of [7]. We borrow the main idea from [7], but our algorithm classifies disks based on their radii, allowing better use of smaller disks when covering the output set of disks by the PTAS for the DUDC problem.

For a given $\epsilon > 0$, we define $\epsilon' = \epsilon/1.944$. Given an instance of the point burning problem, specified by a set $\mathcal{P}$ of n points, we form instances $(\mathcal{P}, (\mathcal{P}, g))$ of the DUDC problem that asks for covering points in $\mathcal{P}$ with a minimum subset of a set of disks centred at $\mathcal{P}$. Unlike the anywhere pointing, additional disks are not added in the DUDC instance, given that all disks must be centred at points of $\mathcal{P}$. As before, we let the radii of these disks take guess values g, and repeatedly apply the PTAS for the DUDC problem with parameter ϵ' to find the smallest guess value h^* for which the PTAS returns a set U of at most h^* disks. With a similar argument as before, the optimal burning number is at least h^*. If h^* is a constant, we exhaustively find an optimal burning scheme in polynomial time. Therefore, we assume h^* is arbitrarily large and let g^* be the smallest value that is no smaller than h^* and is divisible by 10^4; that is, $g^* < h^* + 10^4$.

In what follows, we show that it is possible to cover all points in $\mathcal{P}$ with a set D of $1.944g^*$ disks of radii $\{0, 1, \dots, 1.944g^*\}$. The main challenge is that, unlike anywhere burning, the disks must be centred at points of $\mathcal{P}$. We classify disks in D by their radii. The largest $0.944g^*$ disks from D, each with a radius $\geq g^*$, belong to Class 0, and each is used to cover a disk in U; in total, they cover $0.944g^*$ disks of U. There is no class $1, \dots, 5$. Disks of D with radius in $[0.944g^*, g^*)$ belong to Class 6. We center each disk of Class 6 in one of the $0.056g^*$ disks of U that are not covered by disks of Class 0. At this point, all points in $\mathcal{P}$ are covered, except for those located in the $0.056g^*$ annuli defined by the rings between disks of U (of radius g^*) and disks of Class 6 that partially cover them (of radius $\geq 0.944g^*$).

Let k be any integer such that $7 \leq k \leq 16$. Disks of D with radii in $[2g^* \sin(2\pi/k), 2g^* \sin(2\pi/(k-1)))$ belong to class k. Table 2 shows the exact integer ranges for these classes. Consider the annulus defined by the ring between two co-centred circles of radii g^* and $0.944g^*$. Define a *block-arc* of order k as the partitioning of the annulus into k equal parts via rays that pass the shared center of the two circles. Every circle with a center inside a block-arc of order k and with radius $\geq 2g^* \sin(\pi/k)$ covers that entire black-arc. This is because, in the convex hull of the block-arc, the maximum distance between any two points is realized by the chord of the larger circle, which is of length $2g^* \sin(\pi/n)$. Figure 4 provides an illustration for $k = 7$. Therefore, for any $k \in \{7, 8, \dots, 16\}$, we can partition the annulus (of a disk that is partially covered by disks of Class 6) into k equal block-arcs and cover any "non-empty" block-arc that contains points from $\mathcal{P}$ by a disk of class k centred at any point from $\mathcal{P}$ in the block-arc. By the above observation, all points in the block-arc are covered by

k	radius range	no. disks in the class	partially covered disks from U
1	$[g^*, 1.944g^*]$	$0.944g^*$	$0.944g^*$
6	$[0.944g^*, g^*)$	$0.056g^*$	$0.056g^*$
7	$[0.8678g^*, 0.944g^*)$	$0.0762g^*$	$\lfloor 0.0762g^*/7 \rfloor > 0.01g^*$
8	$[0.7654g^*, 0.8678g^*)$	$0.1024g^*$	$\lfloor 0.1024g^*/8 \rfloor = 0.0128g^*$
9	$[0.6841g^*, 0.7654g^*)$	$0.0813g^*$	$\lfloor 0.0813g^*/9 \rfloor > 0.009g^*$
10	$[0.6181g^*, 0.6841g^*)$	$0.066g^*$	$\lfloor 0.066g^*/10 \rfloor = 0.0066g^*$
11	$[0.5635g^*, 0.6181g^*)$	$0.0546g^*$	$\lfloor 0.0546g^*/11 \rfloor > 0.0049g^*$
12	$[0.5177g^*, 0.5635g^*)$	$0.0458g^*$	$\lfloor 0.0458g^*/12 \rfloor > 0.0038g^*$
13	$[0.4787g^*, 0.5177g^*)$	$0.039g^*$	$\lfloor 0.039g^* \rfloor /13 = 0.003g^*$
14	$[0.4451g^*, 0.4787g^*)$	$0.0336g^*$	$\lfloor 0.0336g^*/14 \rfloor = 0.0024g^*$
15	$[0.4159g^*, 0.4451g^*)$	$0.0292g^*$	$\lfloor 0.0292g^*/15 \rfloor > 0.0019g^*$
16	$[0.3902g^*, 0.4159g^*)$	$0.0257g^*$	$\lfloor 0.0257g^*/16 \rfloor > 0.0016g^*$

Table 2: A summary of classes used in our point burning algorithm. Disks of D in Class 0 cover $0.944g^*$ disk from U. Disks of Class 6 partially cover the remaining $0.56g^*$ disks. The total number of partially covered disks classes $k \in \{7, \dots, 16\}$ is larger than $g^*(0.01+0.0128+0.009+0.0066+0.0049+0.0038+0.003+0.0024+0.0019+0.0016) = 0.056g^*$; that is, disks of classes $k \in \{7, \dots, 16\}$ complete the covering of disks partially covered by Class 6 disks.

such a disk. That is, we can cover the annulus with k disks of class k.

In summary, out of the g^* disks of U, we fully cover $0.944g^*$ disks using Class 0 disks and partially cover the remaining disks with Class 6 disks. The uncovered parts (blocks-arcs) of these disks will be covered by disks of classes $k \in \{7, 16\}$. In doing that, each group of k disks of class k will complete the covering of one disk of U. Table 2 summarizes the number of disks of class k and the number of disks from U that they cover (along with disks of Class 6). We can conclude the following theorem.

Theorem 3 *There is an algorithm with an approximation factor of at most $1.944 + \epsilon$ for the point burning problem for any arbitrary small $\epsilon > 0$.*

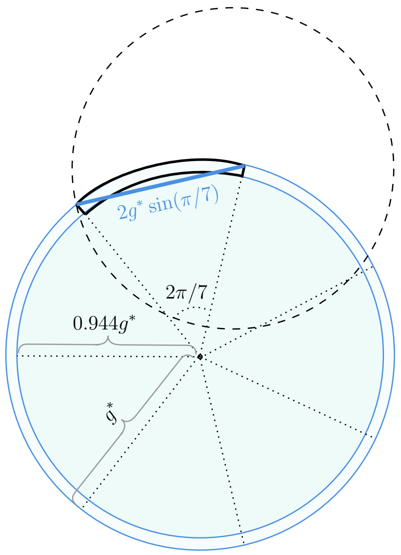


Figure 4: Any disk of radius $2\sin(\pi/k)$ centred at any point in the block-arc covers all points in the block-arc.

Proof. Recall that h^* is the smallest value for which the PTAS for the DUDC instance with parameter $\epsilon' = \epsilon/1.944$ has an output with at most h^* disks; we can conclude the burning number of the input set $\mathcal{P}$ is at least $(h^*-1)/(1+\epsilon')$. Since $h^* > g^* - 10^4$, burning $\mathcal{P}$ requires at least $(g^* - 10^4)/(1+\epsilon')$ rounds.

All points in $\mathcal{P}$ can be covered with the set U of h^* disks of uniform radius h^*, and thus with g^* disks of uniform radius g^*. As discussed above, it is possible to cover all disks in U with $1.944g^*$ disks of radii $\{0, 1, \dots \lfloor 1.944g^* - 1 \rfloor\}$, which ensures a burning scheme that completes within $1.944g^*$ rounds.

The approximation factor of the algorithm is thus at most $\frac{\lfloor 1.944g^* \rfloor}{(g^*-10^4)/(1+\epsilon')}$, which converges to $1.944(1+\epsilon') = 1.944 + \epsilon$ for large values of g^*. □

4 Concluding Remarks

In this paper, we built over the results of Kiel et al. [9] and Gokhale et al. [7] to improve the approximation factor of anywhere burning algorithms from $1.92188+\epsilon$ to $1.865+\epsilon$ and that of point burning algorithms from $1.96296+\epsilon$ to $1.944+\epsilon$. Our improvements come from classifying disks of various radii to cover a set of larger disks of uniform radii given by the PTAS for the Discrete Unit Disc Cover (DUDC) problem. Refining the classification and better "mix-and-match" strategies may result in further gains in the approximation factor. However, it is necessary to go beyond algorithms that solve DUDC instances to achieve notable improvements over the results in this paper.

References

[1] S. Bessy, A. Bonato, J. C. M. Janssen, D. Rautenbach, and E. Roshanbin. Burning a graph is hard. *Discret. Appl. Math.*, 232:73–87, 2017.

[2] A. Bonato. A survey of graph burning. *Contributions Discret. Math.*, 16(1):185–197, 2021.

[3] A. Bonato, J. Janssen, and E. Roshanbin. Burning a graph as a model of social contagion. In *Proc. WAW*, pages 13–22. Springer, 2014.

[4] A. Bonato and S. Kamali. Approximation algorithms for graph burning. In *Proc. TAMC*, pages 74–92, 2019.

[5] E. Friedman. Circles covering circles. `https://erich-friedman.github.io/packing/circovcir/`. Accessed: 2023-04-14.

[6] J. García-Díaz, J. C. Pérez-Sansalvador, L. M. X. Rodríguez-Henríquez, and J. A. Cornejo-Acosta. Burning graphs through farthest-first traversal. *IEEE Access*, 10:30395–30404, 2022.

[7] P. Gokhale, J. M. Keil, and D. Mondal. Improved and generalized algorithms for burning a planar point set. In *Proc. WALCOM*, pages 90–101, 2023.

[8] S. Kamali, A. Miller, and K. Zhang. Burning two worlds. In *Proc. SOFSEM*, pages 113–124. Springer, 2020.

[9] J. M. Keil, D. Mondal, and E. Moradi. Burning number for the points in the plane. In *Proc. CCCG*, pages 205–211, 2022.

[10] R. Kershner. The number of circles covering a set. *American Journal of Mathematics*, 61(3):665–671, 1939.

[11] M. Lieskovskỳ and J. Sgall. Graph burning and non-uniform k-centers for small treewidth. In *Proc. WAOA*, pages 20–35. Springer, 2022.

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

[13] E. H. Neville. On the solution of numerical functional equations, illustrated by an account of a popular puzzle and of its solution. *London Math. Soc.*, 14:308–326, 1915.

[14] E. W. Weisstein. Disk covering problem (from MathWorld–a Wolfram Web Resource). `https://mathworld.wolfram.com/DiskCoveringProblem.html`. Accessed: 2023-04-14.

[15] E. W. Weisstein. Disk covering problem. mathworld–a wolfram web resource. 15(1), 2018.

[16] C. T. Zahn. Black box maximization of circular coverage. *J. Res. Nat. Bur. Stand.*, pages 181–216, 1962.

On the Budgeted Hausdorff Distance Problem

Sariel Har-Peled*   Benjamin Raichel†

Abstract

Given a set P of n points in the plane, and a parameter k, we present an algorithm, whose running time is $O\left(n^{3/2}\sqrt{k}\log^{3/2} n + kn\log^2 n\right)$, with high probability, that computes a subset $Q^\star \subseteq P$ of k points, that minimizes the Hausdorff distance between the convex-hulls of $Q^\star$ and P. This is the first subquadratic algorithm for this problem if k is small.

1 Introduction

Given a set of points P in $\mathbb{R}^d$, a natural goal is to find a small subset of it that represents the point set well. This problem has attracted a lot of interest over the last two decades, and this subset of P is usually referred to as a *coreset* [3, 2]. An alternative approximation is provided by the largest enclosed ellipsoid inside $\mathcal{C}(P)$ (here $\mathcal{C}(P)$ denotes the *convex-hull* of P) or the smallest area bounding box of P (not necessarily axis-aligned). This provides a constant approximation to the projection width of P in any direction v – that is, the projection of P into the line spanned by v is contained in the projection of the ellipsoid after appropriate constant scaling. One can show that in two dimensions, there is a subset $Q \subseteq P$ (i.e., a coreset) of size $O(1/\sqrt{\varepsilon})$ such that the projection width of P and Q is the same up to scaling by $1+\varepsilon$. See Agarwal *et al.* [3, 2] for more details.

The concept of a coreset is attractive as it provides a notion of approximating that adapts to the shape of the point set. However, an older and arguably simpler approach is to require that $\mathcal{C}(Q)$ approximates $\mathcal{C}(P)$ within a certain absolute error threshold. A natural such measure is the ***Hausdorff*** distance between sets $X, Y \subseteq \mathbb{R}^2$, which is

$$\mathsf{d}_H(X,Y) = \max\Bigl(\mathsf{d}(X \to Y),\, \mathsf{d}(Y \to X)\Bigr), \qquad (1)$$

where

$$\mathsf{d}(X \to Y) = \max_{x \in X}\min_{y \in Y}\|xy\|.$$

In our specific case, the two sets are $\mathcal{C}(P)$ and $\mathcal{C}(Q)$, and let $\mathsf{D}_H(Q,P) = \mathsf{d}_H\bigl(\mathcal{C}(Q), \mathcal{C}(P)\bigr)$. The natural questions are

(I) MinCardin: Compute the smallest subset $Q \subseteq P$, such that $\mathsf{D}_H(Q,P) \le \tau$, where τ is a prespecified error threshold. Formally, let

$$\mathcal{F}_{\le\tau} = \left\{ Q \subseteq P \;\middle|\; \mathsf{D}_H(Q,P) \le \tau \right\},$$

and let $k^\star = k^\star(P,\tau) = \min_{Q \in \mathcal{F}_{\le\tau}} |Q|$ denote the minimum cardinality of such a set Q.

(II) MinDist: Compute the subset $Q \subseteq P$ of size k, such that $\mathsf{D}_H(Q,P)$ is minimized, where k is a prespecified subset size threshold. Let $\tau^\star = \tau^\star(P,k) = \min_{Q \subseteq P : |Q| = k} \mathsf{D}_H(Q,P)$ denote the optimal radius.

The two problems are "dual" to each other – solve one, and you get a way to solve the other in polynomial time via a search on the values of the other parameter. In particular, solving both problems directly (in two dimensions) can be done via dynamic programming, but even getting a subcubic running time is not immediate in this case. Indeed, the problem seems to have a surprisingly subtle and intricate structure that make this problem more challenging than it seems at first.

Klimenko and Raichel [7] provided an $O(n^{2.53})$ time algorithm for MinCardin. Very recently, Agarwal and Har-Peled [1] provided a near-linear time algorithm for MinCardin that runs in near linear time if $k^\star = k^\star(P,\tau)$ is small. Specifically, the running time of this algorithm is $O(k^\star n \log n)$.

The purpose of this work is to come up with a subquadratic algorithm for the "dual" problem MinDist. An algorithm with running time $O(n^2 \log n)$ follows readily by computing all possible critical values, and performing a binary search over these values, using the procedure of [1] as a black box. The only subquadratic algorithm known previously was for the special case when P is in convex position, for which [7] gave an algorithm whose running time is $O(n \log^3 n)$ with high probability.

Our main result is an algorithm that, given P and k as input, solves MinDist in $O\left(n^{3/2}\sqrt{k}\log^{3/2} n + kn\log^2 n\right)$ time, with high probability, see Theorem 7 for details. We believe the algorithm itself is technically interesting – it uses random sampling to reduce the range of

*Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; sariel@illinois.edu; http://sarielhp.org/. Work on this paper was partially supported by a NSF AF award CCF-1907400.

†Department of Computer Science; University of Texas at Dallas; Richardson, TX 75080, USA; benjamin.raichel@utdallas.edu; http://utdallas.edu/~benjamin.raichel. Work on this paper was partially supported by NSF CAREER Award 1750780.

interest into an interval containing $O(\sqrt{n})$ critical values. It then use the decision procedure of [1] as a way to compute the critical values in this interval, by "peeling" them one by one in decreasing order. Using random sampling for parametric search is an old idea, see [6] and references there.

2 Preliminaries

Given a point set X in $\mathbb{R}^2$, let $\mathcal{C}(X)$ denote its ***convex hull***. For two compact sets $X, Y \subset \mathbb{R}^2$, let $\mathsf{d}(X,Y) = \min_{x\in X, y\in Y} \|xy\|$ denote their distance. For a single point x let $\mathsf{d}(x,Y) = \mathsf{d}(\{x\},Y)$.

Consider two finite point sets $Q \subseteq P \subset \mathbb{R}^2$, and observe that

$$\mathsf{D}_H(Q,P) = \mathsf{d}_H\big(\mathcal{C}(Q), \mathcal{C}(P)\big) = \max_{p\in P} \mathsf{d}\big(p, \mathcal{C}(Q)\big),$$

see Eq. (1). The first equality above is by definition, and the second is since $Q \subseteq P$ and so we have that $\mathcal{C}(Q) \subseteq \mathcal{C}(P)$, and moreover the furthest point in $\mathcal{C}(P)$ from $\mathcal{C}(Q)$ is always a point in P.

In this paper we consider the following two related problems, where for simplicity, we assume that P is in general position.

Problem 1 (MinCardin) *Given a set $P \subset \mathbb{R}^2$ of n points, and a value $\tau > 0$, find the smallest cardinality subset $Q \subseteq P$ such that $\mathsf{D}_H(Q,P) \le \tau$.*

Problem 2 (MinDist) *Given a set $P \subset \mathbb{R}^2$ of n points, and an integer k, find the subset $Q \subseteq P$ that minimizes $\mathsf{D}_H(Q,P)$ subject to the constraint that $|Q| \le k$.*

For either problem let $Q^\star$ denote an optimal solution. For MinCardin let $k^\star = k^\star(P,\tau) = |Q^\star|$, and for MinDist let $\tau^\star = \tau^\star(P,k) = \mathsf{D}_H(Q^\star, P)$. The algorithms discussed in this paper will output the set $Q^\star$, though when it eases the exposition, we occasionally refer to $k^\star$ as the solution to MinCardin and $\tau^\star$ as the solution to MinDist.

Theorem 3 ([1]) *Given as an input a point set P and parameters k and τ, let $k^\star = k^\star(P,\tau)$. There is a procedure decider(P,τ,k), that in $O(nk\log n)$ time, either returns that "$k^\star > k$", or alternatively returns a set $Q^\star \subseteq P$, such that $|Q^\star| = k^\star \le k$, and $\mathsf{D}_H(Q^\star, P) \le \tau$.*

The above theorem readily implies that the problem MinCardin can be solved in $O(nk^\star \log n)$ time.

Given an input of size n, an algorithm runs in $O(f(n))$ time *with high probability*, if for any chosen constant $c > 0$, there is a constant α_c such that the running time exceeds $\alpha_c f(n)$ with probability $< 1/n^c$.

3 Algorithm

3.1 The canonical set

Given an instance P, k of MinDist, let $Q^\star$ denote an optimal solution. Recall that

$$\tau^\star = \mathsf{D}_H(Q^\star, P) = \max_{p\in P} \mathsf{d}(p, \mathcal{C}(Q^\star)).$$

Assume that $\tau^\star > 0$, which can easily be determined by checking if $|\mathsf{V}(\mathcal{C}(P))| > k$, where $\mathsf{V}(\mathcal{C}(P))$ denotes the set of vertices of $\mathcal{C}(P)$. Let

$$p = \arg\max_{p'\in P} \mathsf{d}(p', \mathcal{C}(Q^\star)),$$

and let q be its projection onto $\mathcal{C}(Q^\star)$, i.e. $\tau^\star = \|pq\|$. Observe that q either lies on a vertex of $\mathcal{C}(Q^\star)$ or in the interior of a bounding edge. Since $Q^\star \subseteq P$, we can conclude that $\tau^\star$ is either (i) the distance between two points in P, or (ii) the distance from a point in P to the line passing through two other points from P. Note that, in case (ii), q must be the orthogonal projection of p on to the line ℓ supporting the edge, and that p must be the furthest point from ℓ out of the points that lie in one of its two defining halfplanes. In particular, for an ordered pair $a, b \in P$ define $\ell_{a,b}$ as the line through a and b, directed from a to b, and let $P_{a,b}$ be the subset of P lying in the halfspace bounded by and to the left of $\ell_{a,b}$. We thus define the following two sets.

$$\mathcal{V} = \big\{ \|xy\| \;\big|\; x, y \in P \big\}$$

and

$$\mathcal{L} = \Big\{ \max_{p\in P_{a,b}} \mathsf{d}(p, \ell_{a,b}) \;\Big|\; a, b \in P \Big\}. \qquad (2)$$

The set $\Xi = \mathcal{V} \cup \mathcal{L}$ is the ***canonical set*** of distance values (i.e., the set of all *critical* values). By the above discussion, we have $\tau^\star \in \Xi$.

Observe that $\mathcal{V}$ and $\mathcal{L}$ (and hence Ξ) have quadratic size. Thus we will not explicitly compute these sets. Instead we will search over $\mathcal{V}$ using the following "median" selection procedure.

Theorem 4 ([4]) *Given a set $P \subset \mathbb{R}^2$ of n points, and an integer $k > 0$, with high probability, in $O(n^{4/3})$ time, one can compute the value of rank k in $\mathcal{V}$.*

For values in $\mathcal{L}$, the algorithm samples values and searches over them, using a procedure loosely inspired by [6]. For that we have the following standard lemma, whose proof we include for completeness.

Lemma 5 ([5]) *Let $P \subset \mathbb{R}^2$ be a set of n points. Then in $O(n\log n)$ time one can build a data structure such that for any query vector $\overrightarrow{u}$, in $O(\log n)$ time, it returns the point of P extremal in the direction $\overrightarrow{u}$, i.e. the point maximizing the dot product with $\overrightarrow{u}$. Let extremal$(\overrightarrow{u})$ denote this query procedure.*

Proof. Let $\mathsf{V}(\mathcal{C}(P)) = \{q_1, \ldots, q_k\}$ be labelled in clockwise order. Let $U(q_i)$ be the set of unit vectors $\overrightarrow{u}$ such that when we translate P so that q_i lies at the origin, then $\overrightarrow{u}$ lies in the exterior angle between the normals of $q_{i-1}q_i$ and q_iq_{i+1}. Observe that **extremal**$(\overrightarrow{u}) = q_i$ precisely when $u \in U(q_i)$. Moreover, the $U(q_i)$ define a partition of the set of all unit vectors into k sets. Thus if we maintain these intervals in an array, sorted in clockwise order, then in $O(\log k) = O(\log n)$ time we can binary search to find which interval $\overrightarrow{u}$ falls in. It takes $O(n \log n)$ time to compute $\mathcal{C}(P)$ and thus the data structure. □

In the next section, given a directed line ℓ, we use the above lemma to make extremal queries for the normal of ℓ lying in its left defining halfplane. This lets us evaluate extreme points for lines supporting edges of the current hull, as well as allows us to sample values from $\mathcal{L}$, for which we have the following.

Corollary 6 *Given a set $P \subset \mathbb{R}^2$ of n points, after $O(n \log n)$ preprocessing time, one can return, in $O(\log n)$ time, a value sampled uniformly at random from $\mathcal{L}$.*

Proof. Sample uniformly at random a pair of points from P, and then use Lemma 5 for the normal to the line passing through this pair of points. □

3.2 The algorithm in stages

The input is a set P of n points, and a parameter k. The task at hand is to compute the minimum distance $\tau^\star$, such that there is a subset $Q \subseteq P$ of size k, such that $\mathsf{D}_H(Q, P) \leq \tau^\star$.

Searching and testing for the optimal value. The algorithm maintains an interval (r, R), such that the following invariants are maintained:

(I) $k^\star(P, r) > k$,

(II) $k^\star(P, R) \leq k$, and

(III) $\tau^\star(P, k) \in (r, R)$.

(The first two conditions are actually implied by the last condition, though for clarity we list all three.) In the following, let $\delta > 0$ denote an infinitesimal[1]. Given a value $\tau \in (r, R)$, one can decide if $\tau = \tau^\star(P, k)$, by running **decider**(P, τ, k) and **decider**$(P, \tau - \delta, k)$, see Theorem 3. If **decider**$(P, \tau - \delta, k)$ returns that $k^\star(P, \tau - \delta) > k$ and $k^\star(P, \tau) = k$ then clearly τ is the desired optimal value. In this case, the algorithm returns this value and stops.

[1]The algorithm can be described without using infinitesimals, but this is somewhat cleaner.

Updating the current interval. After testing if $\tau = \tau^\star(P, k)$ for a value $\tau \in (r, R)$ as described above, if $\tau \neq \tau^\star(P, k)$ then the algorithm can update the current interval. Indeed, if **decider**(P, τ, k) returns that $k^\star(P, \tau) > k$, then the algorithm sets the current interval to (τ, R). Otherwise, **decider**$(P, \tau - \delta, k)$ returned that $k^\star(P, \tau - \delta) \leq k$ and so the algorithm sets the current interval to (r, τ).

Stage I: Handling pairwise distances. The algorithm sets the initial interval to $(0, \infty)$. (Recall as discussed above that we can assume $\tau^\star > 0$.) The algorithm then binary searches over all pairwise distance from $\mathcal{V} = \binom{P}{2}$ by using the distance selection procedure of Theorem 4, in the process repeatedly updating the current interval as described above. If $\tau^\star \in \mathcal{V}$, then the algorithm will terminate when the search considers this value. Otherwise, this search reduces the current interval to two consecutive pairwise distances from $\mathcal{V}$, $r < R$, such that $\tau^\star \in (r, R)$ and the current interval (r, R) contains no pairwise distance of P in its interior.

Stage II: Sampling edge-vertex distances. The algorithm samples a set Π of $O(n^{3/2} \log n)$ values from $\mathcal{L}$, see Eq. (2), using Corollary 6. Let U be the subset of values of Π that lie inside the current interval. The algorithm binary searches over U, repeatedly updating the current interval as described above (by doing median selection so that U's cardinality halves at each iteration). If $\tau^\star \in U$ then the algorithm will terminate when the search considers this value. Otherwise, the search further reduces to the interval to $I' = (r', R')$. (Which as discussed below, with high probability, contains $O(\sqrt{n})$ values from $\mathcal{L}$.)

Stage III: Peeling the critical edge-vertex distances. The algorithm now continues the search on the interval $I' = (r', R')$ and critical values in it, $I' \cap \Xi = I' \cap \mathcal{L}$. In particular, the solution computed by **decider**(P, R', k) is a set $Q \subseteq P$ of size $\leq k$ such that $\mathsf{D}_H(Q, P) \leq R'$. For every edge on the boundary of $\mathcal{C}(Q)$ the algorithm now computes the point from P furthest away from the line supporting the edge (among the points in the halfplane not containing $\mathcal{C}(Q)$), using extremal queries from Lemma 5. Let α be the largest such computed value over all the edges, and observe that $\alpha = \mathsf{D}_H(Q, P)$.[2] If $\alpha < R'$, then $\alpha \geq \tau^\star(P, k)$. The algorithm tests if $\alpha = \tau^\star(P, k)$, and if so it terminates. Otherwise, it must be that the optimal value lies in the interval (r', α). As $\alpha \in (r', R')$ and $\alpha \in \mathcal{L}$, our new interval (r', α) has at

[2]$\mathsf{D}_H(Q, P)$ must be realized at a value from $\mathcal{L}$ as Stage I eliminated $\mathcal{V}$ values, and thus it sufficed to consider furthest distances to the lines supporting edges rather than the edges themselves, since at the maximum such value they must align.

least one less value from $\mathcal{L}$. The algorithm now continues to the next iteration of Stage III.

The case when $\alpha = R'$ (i.e., the higher end of the active interval) is somewhat more subtle. The algorithm calls **decider**$(P, k, \alpha - \delta)$ to compute a set Q' that realizes $k^\star(P, \alpha - \delta)$, where δ is an infinitesimal. Observe that $k^\star(P, \alpha - \delta) \leq k$, as otherwise $\alpha = R'$ was the desired optimal value. Let $\beta = \mathsf{D}_H(Q', P)$, which can be computed in a similar fashion using Q' as α was computed using Q. The algorithm tests if $\beta = \tau^\star(P, k)$, and if so it terminates. Otherwise, by the same reasoning used above for α, we can conclude our new interval (r', β) has at least one fewer value from $\mathcal{L}$, and thus the algorithm continues to the next iteration of Stage III on the interval (r', β).

3.3 Analysis

Correctness. The correctness of the algorithm is fairly immediate given the discussion above. Namely, the algorithm maintains an interval (r, R) with the invariant that $\tau^\star(P, k) \in (r, R)$ (where initially this interval is $(0, \infty)$). In each step of each stage a value $\tau \in (r, R)$ that is either from $\mathcal{V}$ (in Stage I) or from $\mathcal{L}$ (in Stages II and III) is determined. For this value τ we then update the current interval as described above. Namely, we query **decider**(P, τ, k) and **decider**$(P, \tau - \delta, k)$. If these calls return that $k^\star(P, \tau) \leq k$ and $k^\star(P, \tau - \delta) > k$ then $\tau = \tau^\star(P, k)$ and the algorithm terminates. Otherwise, if $k^\star(P, \tau) > k$ the algorithm proceeds on (τ, R), and if $k^\star(P, \tau - \delta) \leq k$ then it proceeds on (r, τ). In either case the interval contains at least one fewer value from Ξ, and thus eventually the algorithm must terminate with the value $\tau^\star(P, k)$.

Running time analysis. In Stage I the algorithm performs a binary search over $\mathcal{V} = \binom{P}{2}$. This is done using the distance selection procedure of Theorem 4, which with high probability takes $O(n^{4/3})$ time to determine each next query value. Each query is answered using the $O(nk \log n)$ time **decider**$(P, \cdot, k)$ from Theorem 3. Thus in total Stage I takes $O\big((n^{4/3} + nk \log n) \log n\big)$ time with high probability. Here, by the union bound, a polynomial number of high probability events (i.e. the events that each call to selection occurs in $O(n^{4/3})$ time), all occur simultaneously with high probability.

In Stage II the algorithm samples $O(n^{3/2} \log n)$ values from $\mathcal{L}$ using the $O(\log n)$ time sampling procedure of Corollary 6. Next, the algorithm binary searches over these values (this time directly), again using **decider**$(P, \cdot, k)$. Thus in total Stage II takes $O(n^{3/2} \log^2 n + nk \log^2 n)$ time.

Stage III begins with some interval (r', R'). Let $X = |\mathcal{L} \cap (r', R')|$. In each iteration of Stage III, for some subset $Q \subseteq P$ of size at most k, the algorithm computes $\alpha = \mathsf{D}_H(Q, P)$. This is done using at most k calls to the $O(\log n)$ query time Lemma 5. (This same step is potentially done a second time for $\beta = \mathsf{D}_H(Q', P)$). Each iteration of Stage III also performs a constant number of calls to **decider**$(P, \cdot, k)$, thus is total one iteration takes $O(k \log n + nk \log n) = O(nk \log n)$ time. As argued above each iteration of Stage III reduces the number of values from $\mathcal{L}$ in the active interval by at least 1, and thus runs for at most X iterations. Thus the total time of Stage III is $O(Xnk \log n)$.

Observe that since Stage II sampled a set Π of $O(n^{3/2} \log n)$ values from the $O(n^2)$ sized set $\mathcal{L}$, the interval between any two consecutive values of Π with high probability has $O(\sqrt{n})$ values from $\mathcal{L}$. As the interval $I' = (r', R')$ returned by Stage II is such an interval, with high probability $X = O(\sqrt{n})$. As the running time of Stage II dominates the running time of Stage I (with high probability), we thus have that with high probability the total time of all stages is

$$
\begin{aligned}
&O(n^{3/2} \log^2 n + (\log n + X) nk \log n) \\
&= O(n^{3/2} \log^2 n + n^{3/2} k \log n + nk \log^2 n) \\
&= O(n^{3/2} (k + \log n) \log n).
\end{aligned}
$$

Slightly improving the running time. Observe that if the algorithm samples $O(nt \log n)$ values in stage II, then with high probability the last two stages take

$$
O\left(nt \log^2 n + \left(\frac{n^2}{nt} + \log n \right) kn \log n \right)
$$

time. Solving for t, we have

$$
nt \log^2 n = (n^2/t) k \log n \implies t^2 = nk / \log n.
$$

Thus, setting $t = \sqrt{nk / \log n}$, and including the running time of stage I, we get the improved high probability running time bound

$$
\begin{aligned}
&O\left(n^{4/3} \log n + nt \log^2 n + \left(\frac{n^2}{nt} + \log n \right) kn \log n \right) \\
&= O\big(n^{3/2} \sqrt{k} \log^{3/2} n + kn \log^2 n \big).
\end{aligned}
$$

In summary, we get the following result.

Theorem 7 *Given an instance of MinDist, consisting of a set $P \subset \mathbb{R}^2$ of n points and an integer k, the above algorithm computes a set $Q^\star \subseteq P$, of size k, that realizes the minimum Hausdorff distance between the convex-hulls of P and $Q^\star$ among all such subsets – that is, $\tau^\star(P, k) = \mathsf{D}_H(P, Q^\star)$. The running time of the algorithm is $O\big(n^{3/2} \sqrt{k} \log^{3/2} n + kn \log^2 n\big)$ with high probability.*

We remark that under the reasonable assumption that $k = O(n / \log n)$ the running time can be stated more simply as $O(n^{3/2} \sqrt{k} \log^{3/2} n)$.

4 Conclusions

The most interesting open problem left by our work is whether one can get a near-linear running time if k is small. Even beating $O(n^{4/3})$ seems challenging. On the other hand, if one is willing to use $2k$ points then a near linear running time is achievable [7]. However, using less than $2k$ points without increasing the Hausdorff distance in near linear time seems challenging.

References

[1] P. K. Agarwal and S. Har-Peled. Computing optimal kernels in two dimensions. In *Proc. 39th Int. Annu. Sympos. Comput. Geom.* (SoCG), LIPIcs, page to appear. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[2] P. K. Agarwal, S. Har-Peled, and K. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, Math. Sci. Research Inst. Pub. Cambridge, New York, NY, USA, 2005.

[3] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51(4):606–635, 2004.

[4] T. M. Chan and D. W. Zheng. Hopcroft's problem, log-star shaving, 2d fractional cascading, and decision trees. *CoRR*, abs/2111.03744, 2021.

[5] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theor. Comput. Sci.*, 27:241–253, 1983.

[6] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, 2014.

[7] G. Klimenko and B. Raichel. Fast and exact convex hull simplification. In M. Bojanczyk and C. Chekuri, editors, *Proc. 41th Conf. Found. Soft. Tech. Theoret. Comput. Sci.* (FSTTCS), volume 213 of *LIPIcs*, pages 26:1–26:17, Wadern, Germany, 2021. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Approximating the Directed Hausdorff Distance

Oliver A. Chubet* Parth M. Parikh† Donald R. Sheehy‡ Siddharth S. Sheth§

Abstract

The Hausdorff distance is a metric commonly used to compute the set similarity of geometric sets. For sets containing a total of n points, the exact distance can be computed naively in $O(n^2)$ time. In this paper, we show how to preprocess point sets individually so that the Hausdorff distance of any pair can then be approximated in linear time. We assume that the metric is doubling. The preprocessing time for each set is $O(n \log \Delta)$ where Δ is the ratio of the largest to smallest pairwise distances of the input. In theory, this can be reduced to $O(n \log n)$ time using a much more complicated algorithm. We compute $(1+\varepsilon)$-approximate Hausdorff distance in $\varepsilon^{-O(d)}n$ time in a metric space with doubling dimension d. The k-partial Hausdorff distance ignores k outliers to increase stability. Additionally, we give a linear-time algorithm to compute directed k-partial Hausdorff distance for all values of k at once with no change to the preprocessing.

1 Introduction

The Hausdorff distance is a metric on compact subsets of a metric space. Let $(X, \mathbf{d})$ be a metric space and let A and B compact subsets of X. The distance from a point $x \in X$ to the set B is $\mathbf{d}(x, B) := \min_{b \in B} \mathbf{d}(x, b)$. The *directed Hausdorff distance* is $\mathbf{d}_h(A, B) := \max_{a \in A} \mathbf{d}(a, B)$, and the (undirected) *Hausdorff distance* is $\mathbf{d}_H(A, B) := \max\{\mathbf{d}_h(A, B), \mathbf{d}_h(B, A)\}$. This definition leads directly to a quadratic time algorithm for finite sets.

We first preprocess the input sets A and B individually into linear-size metric trees, specifically, greedy trees [4]. All the points of a set are stored as leaves in the greedy tree and an internal node serves as an approximation of all the leaves in its subtree. A subset of greedy tree nodes such that every point of the set is a leaf of exactly one node forms an ε-net of the underlying set. The greedy tree can be used to construct ε-nets of the underlying set at different scales. Our algorithms use greedy trees to maintain such nets for either set and track which nodes of B are close to nodes of A. This approach batches the searches and results in fewer distance computations and so, the Hausdorff distance can be computed quickly.

If the input contains a total of n points, then preprocessing takes $\left(\frac{1}{\varepsilon}\right)^{O(d)} n \log \Delta$ time. Here Δ is the *spread* of the input sets, and d is the doubling dimension of the metric space. We present an algorithm that computes a $(1+\varepsilon)$-approximation of the directed Hausdorff distance from A to B in $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)} n$ time after preprocessing. The preprocessing is especially useful when the same sets are involved in multiple distance computations.

One difficulty of working with the Hausdorff distance is its sensitivity to outliers. There are several variations of the Hausdorff distance that reduce the sensitivity to outliers. Among the simplest is a definition where one can ignore up to some number of outliers. The k^{th}*-partial* directed Hausdorff distance is $\mathbf{d}_h^{(k)}(A, B) := \min_{S \in A^{(k)}} \mathbf{d}_h(S, B)$ where $A^{(k)}$ is the set of all subsets of A with k points removed [9].

One might expect that this is a harder problem than computing the directed Hausdorff distance. However, we show that for approximations in low dimensions, this is not the case; the worst-case running time matches that of our algorithm for the standard case. We present an algorithm that computes a $(1+\varepsilon)$-approximation of $\mathbf{d}_h^k(A, B)$ for all values of k in $O(n + \log \Delta)$ time after the same preprocessing as before. That is, the output is a list of $n+1$ approximate values of $\mathbf{d}_h^k(A, B)$ for $k \in \{0, \ldots, n\}$ and this list is produced in linear time.

2 Related Work

We focus on general metrics with finite doubling dimension. In this more general setting, one may not expect a subquadratic algorithm for computing the Hausdorff distance, however, there are classes of metric spaces for which the Hausdorff distance can be computed more quickly. For example, given point sets in the plane, fast nearest neighbor search data structures [1] are used to give an $O(n \log n)$ time algorithm. If one allows for approximate answers, $O(n \log n)$ time algorithms are possible in low-dimensional Euclidean spaces [2].

It is not easy to get an asymptotic improvement on the naive Hausdorff distance algorithm without using an efficient data structure in higher dimensions. In prac-

*Department of Computer Science, North Carolina State University, oachubet@ncsu.edu

†Department of Computer Science, North Carolina State University, pmparikh@ncsu.edu

‡Department of Computer Science, North Carolina State University, don.r.sheehy@gmail.com

§Department of Computer Science, North Carolina State University, sheth.sid@gmail.com

tice, there exist many heuristics to speed up the naive algorithm [3, 14, 18]. Another popular technique is to use a geometric tree data structure. Zhang et al. [19] use octrees to compute the exact Hausdorff distance between 3D point sets. Nutanong et al. [11] present an algorithm to compute the exact Hausdorff distance using R-trees.

Partial Hausdorff distance was first introduced by Huttenlocher et al. [9]. Although there has been considerable interest in this pseudometric, most results are experimental and to the best of our knowledge, a theoretical running time bound does not exist. We give an algorithm to compute approximate partial Hausdorff distance that runs in linear time after preprocessing.

The algorithms presented in this paper maintain both input sets as individual metric trees and traverse them simultaneously. This approach is similar to that of Nutanong et al. [11], but more generally it has been studied in the machine learning literature as dual-tree algorithms [7]. Search problems such as the k-nearest-neighbor search [5], range search [5], and the all-nearest-neighbor search [13] have been explored using dual-tree algorithms. Any all-nearest-neighbor search algorithm where the query and reference sets are different can also be used to compute the directed Hausdorff distance. Ram et al. [13] present an all-nearest-neighbor algorithm that runs in linear time under some strict assumptions about the underlying metrics. Their running time of this algorithm depends on a constant called the degree of bichromaticity. Moreover, this dependence is exponential in the degree of bichromaticity, and high values can result in a poor bound [6].

3 Background

3.1 Doubling Metrics

Let $(X, \mathbf{d})$ be a *metric space*. A *metric ball* is a subset of X, such that $\mathbf{ball}(c, r) := \{x \in X \mid \mathbf{d}(x, c) \leq r\}$, with center $c \in X$ and radius $r \geq 0$. The *spread* Δ of $A \subseteq X$ is the ratio of the diameter to the smallest pairwise distance of points in A. The *doubling dimension* of X, denoted $\dim(X)$, is the smallest real number d such that any metric ball in X can be covered by at most 2^d balls of half the radius. If $\dim(X)$ is bounded then X is a *doubling metric*. The set A is *λ-packed* if $d(a, b) \geq \lambda$ for any distinct $a, b \in A$. The following lemma by Krauthgamer and Lee [10] is a geometric property for packed and bounded sets.

Lemma 1 (Standard Packing Lemma) *If X is a metric space with $\dim(X) = d$ and $Z \subset \mathbf{ball}(x, r)$ for some $x \in X$ is λ-packed then $|Z| \leq \left(\frac{4r}{\lambda}\right)^d$.*

3.2 Greedy Permutations

Let $P = (p_0, ..., p_{n-1})$ be an ordering of n points. The *i^{th}-prefix* of P is the set P_i containing points $p_0, \ldots, p_{i-1}$. The sequence P is a *greedy permutation* if $\mathbf{d}(p_i, P_i) = \mathbf{d}_H(P_i, P)$ for all $i > 0$. If a sequence P is such that $\mathbf{d}_H(P_i, P) \leq \alpha \mathbf{d}(p_i, P_i)$ for all $i > 0$ where $\alpha > 1$ then P is an α-approximate greedy permutation. If $\mathbf{d}(p_i, q) \leq \alpha \mathbf{d}(p_i, P_i)$, then q is an *approximate nearest predecessor*. We denote the distance to the (approximate) nearest predecessor of p_i by ε_{p_i}. Then there is a $\frac{1}{\alpha}\varepsilon_{p_i}$-packing of the points in the prefix P_i.

There is a straightforward algorithm to compute such approximate greedy permutations and the corresponding predecessor mapping [15, 16]. A greedy permutation can be computed in $O(n \log \Delta)$ time for low-dimensional data [8, 15].

3.3 Greedy Trees

A balltree [12] is defined by recursively partitioning compact subsets of a metric space and representing the partitions in a binary tree. For a ball tree on A, every node $\mathbf{a}$ has a center $a \in A$, and a radius r_a. Although multiple nodes can be centered at the same point, all of our algorithms only consider one such node at a time, so the notation is unambiguous.

A greedy tree [4] is a ball tree that can be built on A using the greedy permutation on A and a mapping from points of A to their predecessors in the greedy permutation.[1] The root of the greedy tree is centered at the first point of the greedy permutation. The rest of the tree is constructed incrementally. At all times, there is a unique leaf centered at each of the previously inserted points. For every point p in the permutation, let q denote its predecessor. Create two nodes, one centered at p and one centered at q. Attach these nodes as the right and left children respectively of the leaf centered at q (see Figure 1). The radius of a node is the distance from the center to the farthest leaf in its subtree. Thus, leaves have a radius of zero. The radii of all other nodes can be computed by traversing the subtrees. The radius of a child is never greater than the radius of its parent. If the greedy permutation is α-approximate, then the resulting greedy tree is said to have a parameter α.

The greedy permutation itself requires $O(n \log n)$ time to compute [8]. Given a greedy permutation and the approximate nearest predecessors, the algorithm presented above computes the corresponding greedy tree in $O(n \log \Delta)$ time ($O(n)$ time to build the tree and $O(n \log \Delta)$ time to compute radii). Proof of the following theorem from Chubet et al. [4] and more details on constructing greedy trees are presented in the

[1]The (approximate) nearest predecessors need not be unique, however for the sake of construction, we assume we have chosen one.

appendix.

Theorem 2 *Let G be a greedy tree with $\alpha > 1$. Then the following properties hold:*

1. *The radius of a node $\mathbf{p}$ is bounded, $r_p \leq \frac{\varepsilon_p}{\alpha-1}$.*
2. *Let X be a set of pairwise independent nodes from G. The centers of X are $\frac{(\alpha-1)r}{\alpha}$-packed, where r is the minimum radius of any parent of a node in X.*

4 Approximating Hausdorff Distance

The main input to Hausdorff is a pair of greedy trees G_A and G_B built on sets A and B as well as a list of their nodes in non-increasing order of radius. The input also includes a parameter $\varepsilon > 0$ that determines the approximation factor. The output is the $(1+\varepsilon)$-approximate directed Hausdorff distance from A to B.

4.1 The Setup

The main data structure used is a bipartite graph N on the nodes of G_A and G_B. This graph is called the *neighbor graph* and it satisfies the following invariants:

- Covering Invariant: Every point of $A \cup B$ is a leaf in some vertex of N.
- Neighbor Invariant: if $\mathbf{d}(a, B) = \mathbf{d}(a, b)$ then there is an edge between the nodes containing a and b.

In addition to the graph, we store a local lower bound on $\mathbf{d}(a, B)$ for each point of $a \in A$ that has been added to N. That is, if a vertex $\mathbf{a} \in N$ is a node of G_A, then the local lower bound of $\mathbf{a}$ is defined as,

$$\ell(a) := \min_{\mathbf{b} \in N(\mathbf{a})} \{\mathbf{d}(a, b) - r_b\},$$

where $N(\mathbf{a})$ is the set of neighbors of $\mathbf{a}$ in N.

4.2 The Algorithm

Initialize N to contain the roots of G_A and G_B connected by an edge and then iterate over the input list. For every node $\mathbf{p}$, replace it in N by its children and connect them to the same vertices that were adjacent to $\mathbf{p}$.

We prevent N from becoming too large by pruning edges that are too long. We use the triangle inequality to identify edges whose removal will not violate the neighbor invariant. Let the vertex set of N be $A' \sqcup B'$ at the start of some iteration. An edge between $\mathbf{a}$ and $\mathbf{b}$ can be safely pruned if $\mathbf{d}(a, b') + r_a < \mathbf{d}(a, b) - r_a - r_b$ for some $\mathbf{b}' \in B'$, without violating the neighbor invariant. In Hausdorff, if $\mathbf{p} \in G_A$, then prune long edges adjacent to the newly added vertices and update their lower bounds. Otherwise, $\mathbf{p} \in G_B$ and so, prune edges adjacent to $\mathbf{a}$ for all $\mathbf{a} \in N(\mathbf{p})$ and update $\ell(a)$. A lower bound update is shown in.

We stop once the unprocessed nodes have radii too small to significantly affect the lower bounds. Hausdorff maintains L as the greatest local lower bound as shown in Figure 5. depicts how L is updated when local lower bounds change. If the radius r of the largest node yet to process satisfies $r \leq \frac{\varepsilon}{2}L$, then the algorithm returns L as a $(1+\varepsilon)$-approximation of $\mathbf{d}_h(A, B)$. Now we prove that L is a $(1+\varepsilon)$-approximation of $\mathbf{d}_h(A, B)$ when r is sufficiently small in Hausdorff.

4.3 Correctness of the Hausdorff Algorithm

The following lemma proves that the pruning step maintains the Neighbor Invariant.

Lemma 3 *Let the vertex set of the neighbor graph N be $A' \sqcup B'$ at the start of an iteration of* Hausdorff. *For any $a \in A$, if $\mathbf{d}(a, B) = \mathbf{d}(a, b)$ then there exists an edge $(\mathbf{a}', \mathbf{b}')$ in N such that $a \in \text{Pts}(\mathbf{a}')$ and $b \in \text{Pts}(\mathbf{b}')$.*

Proof. Given $a \in A$, let $b \in B$ such that $\mathbf{d}(a, b) = \mathbf{d}(a, B)$. The neighbor graph maintains a partition of A and B, so there exists $\mathbf{a}' \in A'$ and $\mathbf{b}' \in B'$ such that $a \in \text{Pts}(\mathbf{a}')$ and $b \in \text{Pts}(\mathbf{b}')$. Then, $\mathbf{d}(a, b) \leq \mathbf{d}(a', b') - r_{a'} - r_{b'}$ by the triangle inequality. Suppose in the current iteration we pruned the edge $(\mathbf{a}', \mathbf{b}')$ in N. Then $(\mathbf{a}', \mathbf{b}')$ must satisfy the pruning condition, $\mathbf{d}(a', b'') + r_{a'} < \mathbf{d}(a', b') - r_{a'} - r_{b'}$, for some $\mathbf{b}'' \in B'$. It follows that $\mathbf{d}(a', b'') + r_{a'} < d(a, b)$. Yet, $\mathbf{d}(a, b) \leq r_{a'} + \mathbf{d}(a', b'')$, so this is a contradiction. Therefore, we cannot prune an edge between nodes storing nearest neighbors. □

Lemma 4 *Let r be the radius of the node to be processed in an iteration of* Hausdorff *and let L be the global lower bound. Then, $L \leq \mathbf{d}_h(A, B) \leq L + 2r$. Moreover, if $r \leq (\frac{\varepsilon}{2})L$, then L is a $(1+\varepsilon)$-approximation of $\mathbf{d}_h(A, B)$.*

Proof. Let the vertex set of the neighbor graph N be $A' \sqcup B'$. We first show that the distance from A' to B is at most $L + r$. We know that $\mathbf{d}_h(A', B) \leq \mathbf{d}_h(A', B')$ because $\mathbf{d}(a, B) \leq \mathbf{d}(a, B')$ for any $\mathbf{a} \in A'$. We also know that $r_p \leq r$ for any vertex $\mathbf{p} \in N$. So,

$$\mathbf{d}(a, B') \leq \min_{b \in B'} \{\mathbf{d}(a, b) - r_b + r\} = \ell(a) + r.$$

It follows that $\mathbf{d}(A', B) \leq \max_{\mathbf{a} \in A'} \{\ell(a) + r\} = L + r$. Furthermore, $\mathbf{d}(a, B) \leq \mathbf{d}(a, A') + \mathbf{d}_h(A', B)$, and we know that $\mathbf{d}(a, A') \leq r$. So, $\mathbf{d}(a, B) \leq L + 2r$. Therefore $L \leq \mathbf{d}_h(A, B) \leq L + 2r$. It follows that if $r \leq \frac{\varepsilon}{2}L$ then $\mathbf{d}_h(A, B) \leq (1+\varepsilon)L$. □

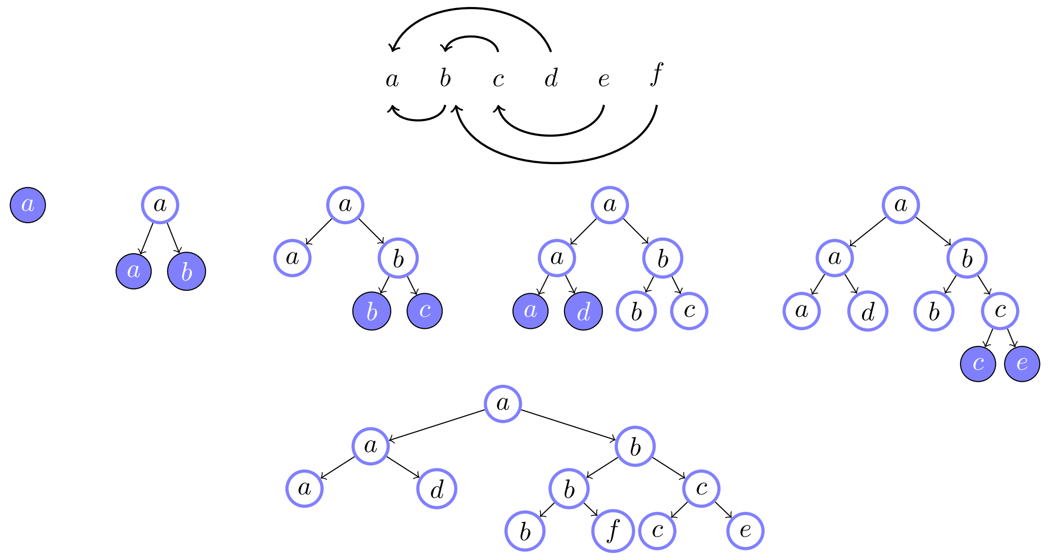


Figure 1: In this figure, a ball tree is computed for a given permutation and predecessor pairing. The permutation (a, b, c, d, e, f) is depicted with arrows representing a predecessor mapping. The tree is constructed incrementally. Each new point creates two new nodes.

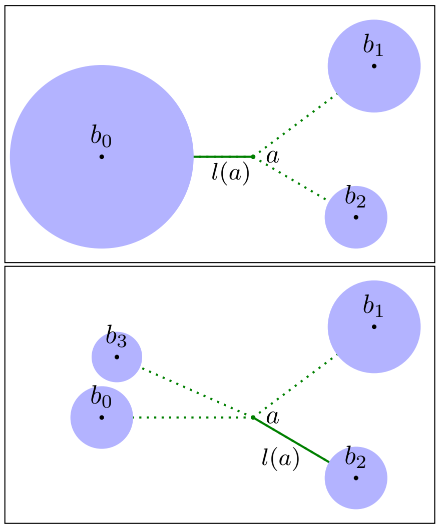


Figure 2: This figure depicts an update of $l(a)$ after replacing a node $\mathbf{b_0}$ with its children.

4.4 Running time of the Hausdorff algorithm

At all times, the degree of every vertex in N is bounded by a constant. We will show that this invariant is guaranteed by the pruning algorithm, the early stopping condition, and a packing bound. It is the critical fact in the following analysis of the Hausdorff running time.

Theorem 5 *Given two greedy trees for sets A and B of total cardinality n,* Hausdorff *computes a $(1+\varepsilon)$-approximation of $\mathbf{d}_h(A,B)$ in $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)} n$ time.*

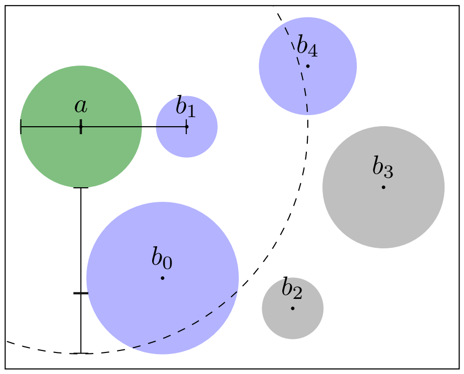


Figure 3: The the nodes $\mathbf{b_2}$ and $\mathbf{b_3}$ are too far away to contain the nearest neighbor of any point in $\mathbf{a}$, so edges $(\mathbf{a}, \mathbf{b_2})$ and $(\mathbf{a}, \mathbf{b_3})$ can be pruned from the neighbor graph. The pruning condition respects the neighbor invariant and does not prune edge $(\mathbf{a}, \mathbf{b_4})$.

Proof. Let G_A and G_B be α-approximate greedy trees. In order to bound the degrees of the neighbor graph N, we first establish that the points associated with the neighbors of a vertex in N are packed. By construction, any node $\mathbf{p} \in N$ is the center of the left- or right-child of a greedy tree node with radius at least r. Then by Theorem 2, $N(\mathbf{p})$ is $\frac{(\alpha-1)r}{\alpha}$-packed. The pruning condition implies that the distance from $\mathbf{p}$ to any of its neighbors

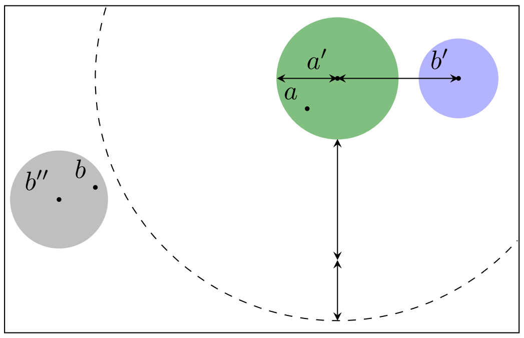


Figure 4: Let $\mathbf{a'} \in A'$. For any point a in $\mathbf{a'}$, by the triangle inequality, $\mathbf{d}(a, B) \leq r_{a'} + \mathbf{d}(a', b')$. If edge $(\mathbf{a}, \mathbf{b''})$ has been pruned, then no point b in $\mathbf{b''}$ can be the nearest neighbor of a, because $\mathbf{d}(a, b') \leq \mathbf{d}(a, b)$ for any such b.

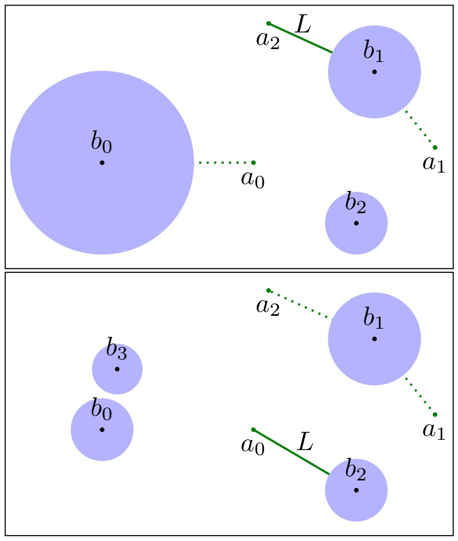


Figure 5: This figure depicts an update of L after replacing the node $\mathbf{b_0}$ with its children. The directed Hausdorff distance, $\mathbf{d}_h(A, B) \geq L$.

is at most $L + 4r$. Thus,

$$|N(\mathbf{p})| \leq \left(\frac{2\alpha(L+4r)}{(\alpha-1)r}\right)^d \leq \left(\frac{16\alpha(1+\varepsilon)}{(\alpha-1)\varepsilon}\right)^d,$$

for all $r \geq (\frac{\varepsilon}{2})L$, by Lemma 1. Therefore, the number of edges incident to any given node in N is $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)}$. So we spend $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)}$ time for each iteration of the algorithm. This gives a running time of $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)} n$. $\square$

5 Approximating Partial Hausdorff Distance

Let A and B be subsets of a metric space $(X, \mathbf{d})$. Let $A^{(k)}$ denote all subsets of A with k elements removed. That is, $A^{(k)} := \{S \subseteq A : |A \backslash S| = k\}$. The k^{th}-*partial* directed Hausdorff distance is $\mathbf{d}_h^{(k)}(A, B) := \min_{S \in A^{(k)}} \mathbf{d}_h(S, B)$. Equivalently, $\mathbf{d}_h^{(k)}(A, B)$ is the $(k+1)^{st}$ largest distance $\mathbf{d}(a, B)$ over all $a \in A$. In particular, $\mathbf{d}_h^{(0)} = \mathbf{d}_h$, as shown in Figure 6.

5.1 Algorithm

The algorithm K-HAUSDORFF approximates the partial directed Hausdorff distance for all $k \leq |A|$ at once. It returns a sequence $(\delta_0, \ldots, \delta_{n-1})$ of distances such that $\delta_i \leq d_h^{(i)}(A, B) \leq (1+\varepsilon)\delta_i$. K-HAUSDORFF builds on HAUSDORFF by replacing the global lower bound L with a max heap of all neighbor graph vertices ordered by their local lower bounds. We call this the *lower bound heap*. Each time the next largest (or approximate largest) distance is found, it is removed from the heap and the algorithm continues as before.

Each time a local lower bound of a node is updated, its priority is also updated. Let λ be the priority of the vertex $\mathbf{m}$ on the top of the heap. When $r \leq \frac{\varepsilon\lambda}{4}$, the node $\mathbf{m}$ is removed from the lower bound heap and the neighbor graph. All the leaves of $\mathbf{m}$ have a distance at most $(1+\varepsilon)\lambda$ from B, so λ is appended to the output sequence once for each leaf.

5.2 Lower Bound Heap

Using a standard heap would require $O(\log n)$ time for basic operations. Allowing for a small approximation, we can put the nodes in buckets, where the m^{th} bucket of this node heap contains nodes with radii in the interval $(\beta^m, \beta^{m+1}]$, where $\beta = 1 + \frac{\varepsilon}{2}$. There are at most $O(\log \Delta)$ buckets. Using a standard heap on the buckets would lead to $O(\log \log \Delta)$ time for basic operations. This can be improved still further using a bucket queue [17], an array of the $O(\log \Delta)$ buckets. Most operations will take constant time except for remove max, which can require iterating through the buckets. We will show that the buckets are removed in non-increasing order so the iteration visits each bucket at most once, incurring a total cost of $O(\log \Delta)$.

In K-HAUSDORFF, the running time of each iteration is now determined by the cost of lower bound updates The local lower bounds are not guaranteed to be non-increasing, so we describe a procedure that allows us to use the bucket queue. Let $s = \left\lceil \log_\beta(\frac{4r\beta}{\varepsilon}) \right\rceil$. Each time the radius decreases we update s and traverse the array until we reach the new bucket s. For any occupied bucket j that we encounter before or coinciding with bucket s, we return $\delta_i = \beta^{\lfloor \log_\beta \ell(p_i) \rfloor}$ for every leaf in

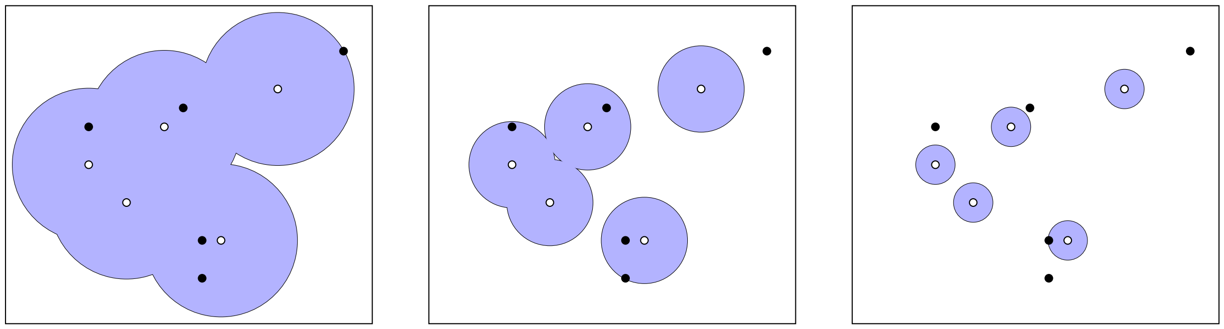


Figure 6: Depicted above are the directed Hausdorff distance $\mathbf{d}_h(A, B)$ (left), the first partial Hausdorff distance $\mathbf{d}_h^{(1)}(A, B)$ (center), and the 4th-partial Hausdorff distance $\mathbf{d}_h^{(4)}(A, B)$ (right).

each $\mathbf{p}_i$ in bucket j. In other words, we remove and return a value all the points whose lower bounds are greater than β^s.

5.3 Correctness and Running Time

The analysis of the k-Hausdorff algorithm closely follows the analysis of the Hausdorff algorithm. As points are removed, the Hausdorff distance decreases, allowing the neighbor graph to maintain a constant degree throughout as in Theorem 5. The main difference in the running time analysis is that one must include the cost of the heap operations.

Theorem 6 *If $\beta = (1 + \frac{\varepsilon}{2})$, then* k-Hausdorff *computes a $(1 + \varepsilon)$-approximation of all partial Hausdorff distances in $(\frac{1+\varepsilon}{\varepsilon})^{O(d)} n + O(\log \Delta)$ time.*

Proof. First we show the correctness of the sequence $(\delta_0, \ldots, \delta_{n-1})$ returned by k-Hausdorff, i.e., $\delta_i \le \mathbf{d}_h^{(i)}(A, B) < (1 + \varepsilon)\delta_i$ for all i. Let L be the global lower bound when δ_i is returned. Then, $L \le \beta\delta_i$ and so, $L \le (1 + \frac{\varepsilon}{2})\delta_i$. Furthermore, $r \le \frac{\varepsilon}{4}\delta_i$ by the stopping condition described in Section 5.2.

For each j, let p_j denote the j^{th} removed point. Let $S_i = A \setminus \{p_j \mid j \le i\}$. It follows by the definition of partial Hausdorff distance that, $\mathbf{d}_h^{(i)}(A, B) \le \mathbf{d}_h(S_i, B)$. Then,

$$\begin{aligned}\mathbf{d}_h^{(i)}(A, B) &\le \mathbf{d}_h(S_i, B)\\ &\le L + 2r\\ &\le \left(1 + \frac{\varepsilon}{2}\right)\delta_i + \frac{\varepsilon}{2}\delta_i\\ &= (1 + \varepsilon)\delta_i.\end{aligned}$$

For the other direction, it is sufficient to show that $\delta_i \le \mathbf{d}_h^{(i)}(A, B)$. Suppose that $\mathbf{d}_h^{(i)}(A, B) < \delta_i$. Then, there exists some set $S \subset A$ such that $|S| = |S_i|$ and $\mathbf{d}_h(S, A) = \mathbf{d}_h^{(i)}(A, B)$. For all $a \in S$, we have $\ell(a) \le \mathbf{d}(a, B) < \delta_i$. By our choice of s and β the δ_i must be non-increasing. Therefore, none of the points in S could have been removed. It follows that $S = S_i$ and this is a contradiction. Therefore, $\delta_i \le \mathbf{d}_h^{(i)}(A, B) \le (1 + \varepsilon)\delta_i$.

Now we analyze the running time of k-Hausdorff. In every iteration k-Hausdorff performs some operations on the neighbor graph and some operations on the lower bound heap. We show that the degrees of nodes in N remain constant as in Theorem 5. By Theorem 2, the neighbor graph is $\frac{(\alpha-1)r}{\alpha}$-packed. Let $\mathbf{p} \in N$. Then by Lemma 1 and the pruning condition,

$$|N(\mathbf{p})| \le \left(\frac{2\alpha(L + 4r)}{(\alpha - 1)r}\right)^d \le \left(\frac{8\alpha(1 + \varepsilon)}{(\alpha - 1)\varepsilon}\right)^d,$$

which is $(\frac{1+\varepsilon}{\varepsilon})^{O(d)}$. Let the local lower bound of point p change from ℓ_p to ℓ'_p in an iteration. We know that,

$$\ell'_p \le \ell_p + r \le \beta^s + \frac{\beta^{s-1}\varepsilon}{4}.$$

Therefore,

$$\lfloor \log_\beta \ell'_p \rfloor \le \left\lfloor \log_\beta \left(\beta^{s-1}\left(\beta + \frac{\varepsilon}{4}\right)\right)\right\rfloor.$$

If $\beta = (1 + \frac{\varepsilon}{2})$, then $\lfloor \log_\beta \ell'_p \rfloor \le s$. So, all partial distances can be computed in a single traversal of the lower bound heap. Therefore, traversing the lower bound heap takes $O(\log \Delta)$ time. Thus the total running time is $(\frac{1+\varepsilon}{\varepsilon})^{O(d)} n + O(\log \Delta)$. □

6 Conclusion

We presented an algorithm to compute the directed Hausdorff distance that runs in $O(n)$ time after computing the greedy trees. The benefits of preprocessing outweigh the costs if the same sets are involved in multiple distance computations. With some modifications, the same algorithm can be used to compute all k-partial Hausdorff distances in $O(n + \log \Delta)$ time.

References

[1] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13(3-4):251–265, September 1995.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, November 1998.

[3] Y. Chen, F. He, Y. Wu, and N. Hou. A local start search algorithm to compute exact Hausdorff Distance for arbitrary point sets. *Pattern Recognition*, 67:139–148, July 2017.

[4] O. Chubet, P. Parikh, D. R. Sheehy, and S. Sheth. Proximity search in the greedy tree. In *2023 Symposium on Simplicity in Algorithms (SOSA)*, pages 332–342.

[5] R. Curtin, W. March, P. Ram, D. Anderson, A. Gray, and C. Jr. Tree-independent dual-tree algorithms. *30th International Conference on Machine Learning, ICML 2013*, 04 2013.

[6] R. R. Curtin, D. Lee, W. B. March, and P. Ram. Plug-and-play dual-tree algorithm runtime analysis. *Journal of Machine Learning Research*, 16(101):3269–3297, 2015.

[7] A. Gray and A. Moore. N-Body Problems in Statistical Learning. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[8] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[9] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.

[10] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 798–807, USA, 2004. Society for Industrial and Applied Mathematics.

[11] S. Nutanong, E. H. Jacox, and H. Samet. An incremental Hausdorff distance calculation algorithm. *Proceedings of the VLDB Endowment*, 4(8):506–517, May 2011.

[12] S. M. Omohundro. Five balltree construction algorithms. Technical Report 562, ICSI Berkeley, 1989.

[13] P. Ram, D. Lee, W. March, and A. Gray. Linear-time algorithms for pairwise statistical problems. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

[14] J. Ryu and S.-i. Kamata. An efficient computational algorithm for Hausdorff distance based on points-ruling-out and systematic random sampling. *Pattern Recognition*, 114:107857, June 2021.

[15] D. R. Sheehy. greedypermutations. https://github.com/donsheehy/greedypermutation, 2020.

[16] D. R. Sheehy. One hop greedy permutations. In *Proceedings of the 32nd Canadian Conference on Computational Geometry*, 2020.

[17] S. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag London Ltd, 2008.

[18] A. A. Taha and A. Hanbury. An Efficient Algorithm for Calculating the Exact Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(11):2153–2163, November 2015.

[19] D. Zhang, F. He, S. Han, L. Zou, Y. Wu, and Y. Chen. An efficient approach to directly compute the exact Hausdorff distance for 3D point sets. *Integrated Computer-Aided Engineering*, 24(3):261–277, July 2017.

Appendix: Greedy Trees

Greedy Permutations

Let $P = (p_0, \ldots, p_{n-1})$ be a finite sequence of points in a metric space with distance $\mathbf{d}$. The i^{th}*-prefix* is the set $P_i = \{p_0, \ldots, p_{i-1}\}$ containing the first i points of P.

The sequence P is a *greedy permutation* if for all i,

$$\mathbf{d}(p_i, P_i) = \max_{p \in P} \mathbf{d}(p, P_i).$$

If a sequence P is such that,

$$\mathbf{d}(p_i, P_i) \leq \alpha \max_{p \in P} \mathbf{d}(p, P_i),$$

for all $i > 0$ where $\alpha > 1$ then P is an α*-approximate greedy permutation*. The point $p_0 \in P$ is the *seed* of the greedy permutation. The *predecessor mapping* $T : P \setminus \{p_0\} \to P$ maps each point p_i in P (other than the seed) to the (approximately) closest point in the prefix P_i. The *insertion distance* of p is $\mathbf{d}(p, T(p))$, denoted ε_p. For α-approximate greedy permutations, we require that the predecessor mapping satisfies the condition that

$$\varepsilon_p \leq \frac{1}{\alpha} \varepsilon_{T(p)}.$$

Constructing Greedy Trees

A *ball tree* [12] on A is a binary tree defined by recursively partitioning A. Each node of the tree stores a center and a radius that covers the points in its leaves. A ball tree *node* $\mathbf{x}$ is centered at $x \in A$ and has a radius r_x. The set of all points contained in the leaves of $\mathbf{x}$ is denoted by $\mathrm{Pts}(\mathbf{x})$. Two nodes $\mathbf{y}$ and $\mathbf{z}$ are *independent* if $\mathrm{Pts}(\mathbf{y}) \cap \mathrm{Pts}(\mathbf{z}) = \varnothing$.

We construct the *greedy tree* as a ball tree from a greedy permutation. The construction applies more generally to any permutation on a point set P with a predecessor mapping T. Given P and T, the *descendants* of a point p are defined recursively so that q is a descendant of p if $q = p$ or $T(q)$ is a descendant of p.

To build a ball tree, it suffices to describe how to partition a set and pick the centers of the two parts. For a set $S \subseteq P$, let a and b be the first and second points of S with respect to their ordering in P. These will be the centers of the two sets. The set centered at b will contain the descendants of b and the set centered at a will contain the rest of the points. The convention is that the left child is centered at a and the right child is centered at b.

The definition in terms of partitions is simple to state, but there is an equivalent definition that is simpler to implement. One can construct the ball tree G for P and T incrementally as follows. Start with a root node centered at p_0. Iterate through the points b in the permutation (starting at p_1). Let $a = T(b)$ and let $\mathbf{x}$ be the unique leaf of G centered at a. Create new nodes centered at a and b and assign them to be the left and right children of $\mathbf{x}$. See Figure 1.

Once the tree is built, the radius of each node can be computed as the distance to its farthest descendant. The number of leaves in the subtree rooted at any node is also stored in the node. This can be used for range counting and k nearest neighbor search.

We say G has a parameter α when P is an α-approximate greedy permutation. As noted previously, for all nodes p_i (other than the root), we construct the greedy permutation so that $\varepsilon_{p_i} \leq \alpha \mathbf{d}(p_i, P_i)$ and $\varepsilon_{T(p_i)} \geq \alpha \varepsilon_{p_i}$.

Structure Theorem

Here we present some properties of the greedy tree that allow us to bound the degree of a vertex in the neighbor graph at any stage of Hausdorff.

Theorem 2 *Let G be a greedy tree with $\alpha > 1$. Then the following properties hold:*

1. *The radius of a node $\mathbf{p}$ is bounded, $r_p \leq \frac{\varepsilon_p}{\alpha-1}$.*
2. *Let X be a set of pairwise independent nodes from G. The centers of X are $\frac{(\alpha-1)r}{\alpha}$-packed, where r is the minimum radius of any parent of a node in X.*

Proof. 1. Let $\mathbf{p}$ be a node in G and $q \in \mathrm{Pts}(\mathbf{p})$ be such that $r_p = \mathbf{d}(p, q)$. Additionally, let $\mathbf{x}$ and $\mathbf{y}$ be nodes containing q such that $T(x) = p$ and $T(y) = x$. By the triangle inequality,

$$r_p \leq \mathbf{d}(p, x) + \mathbf{d}(x, q) \leq \varepsilon_x + r_x.$$

Moreover, by construction, $\varepsilon_x \leq \frac{1}{\alpha}\varepsilon_p$, so $r_p \leq \frac{1}{\alpha}\varepsilon_p + r_x$, so by induction on the height,

$$r_p \leq \sum_{i=1}^{h} \varepsilon_p \left(\frac{1}{\alpha}\right)^i \leq \sum_{i=0}^{\infty} \varepsilon_p \left(\frac{1}{\alpha}\right)^{i+1} \leq \frac{\varepsilon_p}{\alpha - 1}.$$

2. Let X be a set of pairwise independent nodes from G, and let $\mathbf{x}$ be an arbitrary node in X with a parent $\mathbf{p}$. Then, $r_p \leq \varepsilon_x + r_x \leq \varepsilon_x + \frac{\varepsilon_x}{\alpha-1}$. It follows that $\varepsilon_x \geq \frac{\alpha-1}{\alpha} r$ and therefore, H is $\frac{(\alpha-1)r}{\alpha}$-packed, where r is the minimum radius of any parent of a node in X.

□

Convex Hulls and Triangulations of Planar Point Sets on the Congested Clique

Jesper Jansson[*]   Christos Levcopoulos[†]   Andrzej Lingas[‡]

Abstract

We consider geometric problems on planar n^2-point sets in the congested clique model. Initially, each node in the n-clique network holds a batch of n distinct points in the Euclidean plane given by $O(\log n)$-bit coordinates. In each round, each node can send a distinct $O(\log n)$-bit message to each other node in the clique and perform unlimited local computations. We show that the convex hull of the input n^2-point set can be constructed in $O(\min\{h, \log n\})$ rounds, where h is the size of the hull, on the congested clique. We also show that a triangulation of the input n^2-point set can be constructed in $O(\log^2 n)$ rounds on the congested clique.

1 Introduction

The communication/computation model of congested clique focuses on the communication cost and ignores that of local computation. It can be seen as a reaction to the criticized model of Parallel Random Access Machine (PRAM), studied in the 80s and 90s, which focuses on the computation cost and ignores the communication cost [1].

In recent decades, the complexity of dense graph problems has been intensively studied in the congested clique model. Typically, each node of the clique network initially represents a distinct vertex of the input graph and knows that vertex's neighborhood in the input graph. Then, in each round, each of the n nodes can send a distinct message of $O(\log n)$ bits to each other node and can perform unlimited local computation. Several dense graph problems, for example, the minimum spanning tree problem, have been shown to admit $O(1)$-round algorithms in the congested clique model [10]. Note that when the input graph is of bounded degree, each node can send its whole information to a distinguished node in $O(1)$ rounds. The distinguished node can then solve the graph problem locally. However, when the input graph is dense such a trivial solution requires $\Omega(n)$ rounds.

Researchers have also successfully studied problems not falling in the category of graph problems, like matrix multiplication [3] or sorting and routing [6], in the congested clique model. In both cases, one assumes that the basic items, i.e., matrix entries or keys, respectively, have $O(\log n)$ bit representations and that each node initially has a batch of n such items. As in the graph case, each node can send a distinct $O(\log n)$-bit message to each other node and perform unlimited computation in every round. Significantly, it has been shown that matrix multiplication admits an $O(n^{1-2/\omega})$-round algorithm [3], where ω is the exponent of fast matrix multiplication, while sorting and routing admit $O(1)$-round algorithms [6] under the aforementioned assumptions.

We extend this approach to include basic geometric problems on planar point sets. These problems are generally known to admit polylogarithmic time solutions on PRAMs with a polynomial number of processors [1]. Initially, each node of the n-clique network holds a batch of n points belonging to the input set S of n^2 points with $O(\log n)$-bit coordinates in the Euclidean plane. As in the graph, matrix, sorting, and routing cases, in each round, each node can send a distinct $O(\log n)$-bit message to each other node and perform unlimited local computations. Analogously, trivial solutions consisting in gathering the whole data in a distinguished node require $\Omega(n)$ rounds.

First, we provide a simple implementation of the Quick Convex Hull algorithm [9], showing that the convex hull of S can be constructed in $O(h)$ rounds on the congested clique, where h is the size of the hull. Then, we present and analyze a more refined algorithm for the convex hull of S on the congested clique running in $O(\log n)$ rounds. Finally, we present a divide-and-conquer method for constructing a triangulation of S in $O(\log^2 n)$ rounds on the congested clique.

2 Preliminaries

For a positive integer r, $[r]$ stands for the set of positive integers not exceeding r.

Let $S = \{p_1, ..., p_n\}$ be a set of n distinct points in the Euclidean plane such that the x-coordinate of each point is not smaller than that of p_1 and not greater than that of p_n. The *upper hull* of S (with respect to (p_1, p_n)) is the part of the convex hull of S beginning in

[*]Graduate School of Informatics, Kyoto University, Kyoto, Japan, jj@i.kyoto-u.ac.jp

[†]Department of Computer Science, Lund University, 22100 Lund, Sweden, Christos.Levcopoulos@cs.lth.se

[‡]Department of Computer Science, Lund University, 22100 Lund, Sweden, Andrzej.Lingas@cs.lth.se

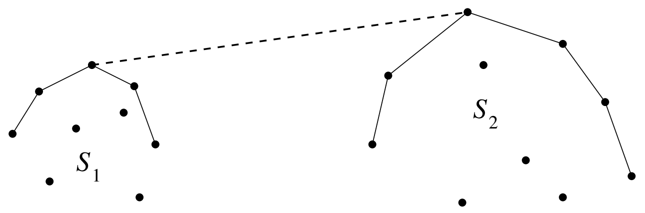


Figure 1: An example of the bridge between the upper hulls of S_1 and S_2.

p_1 and ending in p_n in clockwise order. Symmetrically, the *lower hull* of S (with respect to (p_1, p_n)) is the part of the convex hull of S beginning in p_n and ending in p_1 in clockwise order. A *supporting line* for the convex hull or upper hull or lower hull of a finite point set in the Euclidean plane is a straight line that touches the hull without crossing it properly.

Let S_1, S_2 be two finite sets of points in the Euclidean plane separated by a vertical line. The *bridge* between the upper (or lower) hull of S_1 and the upper (or, lower, respectively) hull of S_2 is a straight line that is a supporting line for the both upper (lower, respectively) hulls. See Figure 1 for an illustration.

3 Quick Convex Hull Algorithm on Congested Clique

The Quick Convex Hull Algorithm is well known in the literature, see, e.g, [9]. Roughly, we shall implement it as follows in the congested clique model. First, the set S of n^2 input points with $O(\log n)$-bit coordinates is sorted by their x-coordinates [6]. As a result, each consecutive clique node gets a consecutive n-point fragment of the sorted S. Next, each node informs all other nodes about its two extreme points along the x axis. By using this information, each node can determine the same pair of extreme points p_{min}, p_{max} in S along the x axis. Using this extreme pair, each node can decompose its subsequence of S into the upper-hull subsequence consisting of the points that lie above or on the segment (p_{min}, p_{max}) and the lower-hull subsequence consisting of points that lie below or on (p_{min}, p_{max}). From now on, the upper hull of S and the lower hull of S are computed separately by calling the procedures $QuickUpperHull(p_{min}, p_{max})$ and $QuickLowerHull(p_{min}, p_{max})$, respectively. The former procedure proceeds as follows. Each node sends a point q of highest y-coordinate among those in its upper-hull subsequence different from p_{min} and p_{max} to all other nodes. Then, each node selects the same point q of maximum y-coordinate among all points in the whole upper-hull subsequence different from p_{min} and p_{max}. Note that q must be a vertex of the upper hull of S. Two recursive calls $QuickUpperHull(p_{min}, q)$ and $QuickUpperHull(q, p_{max})$ follow, etc. The procedure $QuickLowerHull$ is defined symmetrically. As each non-leaf call of these two procedures results in a new vertex of the convex hull, and each step of these procedures but for the recursive calls takes $O(1)$ rounds, the total number of rounds necessary to implement the outlined variant of Quick Convex Hull algorithm, specified in the procedure $QuickConvexHull(S)$, is proportional to the size of the convex hull of S.

procedure $QuickConvexHull(S)$
Input: A set of n^2 points in the Euclidean plane with $O(\log n)$ bit coordinates, each node holds a batch of n input points.
Output: The vertices of the convex hull of S held in clockwise order in consecutive nodes in batches of at most n vertices.

1. Sort the points in S by their x-coordinates so each node receives a subsequence consisting of n consecutive points in S, in the sorted order.

2. Each node sends the first point and the last point in its subsequence to the other nodes.

3. Each node computes the same point p_{max} of the maximum x-coordinate and the same point p_{min} of the minimum x-coordinate in the whole input sequence S based on the gathered information. Next, it decomposes its sorted subsequence into the upper hull subsequence consisting of points above or on the segment connecting p_{max} and p_{min} and the lower hull subsequence consisting of the points lying below or on this segment. In particular, the points p_{min} and p_{max} are assigned to both upper and lower hull subsequences of the subsequences they belong to.

4. Each node sends its first and last point in its upper hull subsequence as well as its first and last point in its lower hull subsequence to all other nodes.

5. $QuickUpperHull(p_{min}, p_{max})$

6. $QuickLowerHull(p_{min}, p_{max})$

7. By the previous steps, each node keeps consecutive pieces (if any) of the upper hull as well as the lower hull. However, some nodes can keep empty pieces. In order to obtain a more compact output representation in batches of n consecutive vertices of the hull (but for the last batch) assigned to consecutive nodes of the clique, the nodes can count the number of vertices on the upper and lower hull they hold and send the information to the other nodes. Using the global information, they can design destination addresses for their vertices on both hulls. Then, the routing protocol from [6] can be applied.

procedure $QuickUpperHull(p, r)$
Input: The upper-hull subsequence of the input point set S held in consecutive nodes in batches of at most n points and two distinguished points p, r in the subsequence , where the x-coordinate of p is smaller than that of r.
Output: The vertices of the upper hull of S with x-coordinates between those of p and r held in clockwise order in consecutive nodes, between those holding p and r respectively, in batches of at most n points.

1. Each node u determines the set S_u of points in its upper-hull subsequence that have x-coordinates between those of p and r and lie above or on the segment between p and r. If S_u is not empty then the node sends a point in S_u having the largest y-coordinate to the clique node holding p, from here on referred to as the *master node*.

2. If the master node has not received any point satisfying the requirements from the previous step then it proclaims p and r to be vertices of the upper hull by sending this information to the nodes holding p and/or q, respectively. (In fact one of the vertices p and r has been marked as being on the upper hull earlier.) Next, it pops a call of $QuickUpperHull$ from the top of a stack of recursive calls held in a prefix of the clique nodes numbered 1, 2,.... In case the stack is empty it terminates $QuickUpperHull(p_{min}, p_{max})$.

3. If the master has received some points satisfying the requirements from Step 1 than it determines a point q of maximum y-coordinate among them. Next, it puts the call of $QuickUpperHull(q, r)$ on the top of the stack and then activates $QuickUpperHull(p, q)$.

The procedure $QuickLowerHull(p, r)$ is defined analogously.

Each step of the procedure $QuickConvexHull(S)$, but for the calls to $QuickUpperHull(p_{min}, p_{max})$ and $QuickLowerHull(p_{min}, p_{max})$ can be done in $O(1)$ rounds on the congested clique on n nodes. In particular, the sorting and the routing steps in $QuickConvexHull(S)$ can be done in $O(1)$ rounds by [6]. Similarly, each step of $QuickUpperHull(p, r)$, and symmetrically each step of $QuickLowerHull(p, r)$, but for recursive calls, can be done in $O(1)$ rounds. Since each non-leaf (in the recursion tree) call of $QuickUpperHull(p, r)$ and $QuickLowerHull(p, r)$ results in a new vertex of the convex hull, their total number does not exceed h. Hence, we obtain the following theorem.

Theorem 1 *Consider a congested n-clique network, where each node holds a batch of n points in the Euclidean plane specified by $O(\log n)$-bit coordinates. Let h be the number of vertices on the convex hull of the set S of the n^2 points. The convex hull of S can be computed by the procedure $QuickConvexHull(S)$ in $O(h)$ rounds on the congested clique.*

4 An $O(\log n)$-round Algorithm for Convex Hull on Congested Clique

Our refined algorithm for the convex hull of the input point set S analogously as $QuickConvexHull(S)$ starts by sorting the points in S by their x-coordinates and then splitting the sorted sequence of points in S into an upper-hull subsequence and lower-hull subsequence. Next, it computes the upper hull of S and the lower hull of S by calling the procedures $NewUpperHull(s)$ and $NewLowerHull(S)$, respectively. The procedure $NewUpperHull(S)$ lets each node ℓ construct the upper hull H_ℓ of its batch of at most n points in the upper-hull subsequence locally. The crucial step of $NewUpperHull(S)$ is a parallel computation of bridges between all pairs H_ℓ, H_m, $\ell \neq m$, of the constructed upper hulls by parallel calls to the procedure $Bridge(H_\ell, H_m)$. Based on the bridges between H_ℓ and the other upper hulls H_m, each node ℓ can determine which of the vertices of H_ℓ belong to the upper hull of S (see Lemma 1). The procedure $Bridge$ has recursion depth $O(\log n)$ and the parallel implementation of the crucial step of $NewUpperHull(s)$ takes $O(\log n)$ rounds. The procedure $NewLowerHull(s)$ is defined symmetrically. Consequently, the refined algorithm for the convex hull of S specified by the procedure $NewConvexHull(S)$ can be implemented in $O(\log n)$ rounds.

The procedure $NewConvexHull(S)$ is defined in exactly the same way as $QuickConvexHull(S)$, except that the calls to $QuickUpperHull(p_{min}, p_{max})$ and $QuickLowerHull(p_{min}, p_{max})$ are replaced by calls to $NewUpperHull(S)$ and $NewLowerHull(S)$, respectively.

procedure $NewUpperHull(S)$
Input: The upper-hull subsequence of the input point set S held in consecutive nodes in batches of at most n points.
Output: The vertices of the upper hull of S held in clockwise order in consecutive nodes in batches of at most n vertices.

1. Each node ℓ computes the upper hull H_ℓ of its upper-hull subsequence locally.

2. In parallel, for each pair ℓ, m of nodes, the procedure $Bridge(H_\ell, H_m)$ computing the bridge between H_ℓ and H_m is called. (The procedure uses the two nodes in $O(\log n)$ rounds, exchanging at most two messages between the nodes in each of these rounds.)

3. Each node ℓ checks if it has a single point p not marked as not qualifying for the upper hull of S such that there are bridges between H_k and H_ℓ and H_ℓ and H_m, where $k < \ell < m$, p is an endpoint of both bridges, and the angle formed by the two bridges is smaller than 180 degrees. If so, p is also marked as not qualifying for the upper hull of S.

4. Each node ℓ prunes the set of vertices of H_l, leaving only those vertices that have not been marked in the previous steps (including calls to the procedure *Bridge*) as not qualifying for the upper hull of S.

The following lemmata enable the implementation of the n^2 calls to $Bridge(H_\ell, H_m)$ in the second step of $NewUpperHull(S)$ in $O(\log n)$ rounds on the congested clique.

Lemma 2 *For $\ell \in [n]$, let H_ℓ be the upper hull of the upper-hull subsequence of S assigned to the node ℓ. A vertex v of H_ℓ is not a vertex of the upper hull of S if and only if it lies below a bridge between H_ℓ and H_m, where $\ell \neq m$, or there are two bridges between H_ℓ and H_s, H_t, respectively, where $s < \ell < t$, such that they touch v and form an angle of less than 180 degrees at v.*

Proof. Clearly, if at least one of the two conditions on the right side of "if and only if" is satisfied then v cannot be a vertex of the upper hull of S. Suppose that v is not a vertex of the upper hull of S. Then, since it is a vertex of H_l, there must be an edge e of the upper hull of S connecting H_k with H_m for some $k \leq \ell \leq m$, $k \neq m$, that lies above v. We may assume without loss of generality that v does not lie below any bridge between H_ℓ and H_q, $\ell \neq q$. It follows that $s < \ell < t$. Let b_k be the bridge between H_k and H_ℓ, and let b_m be the bridge between H_ℓ and h_m. It also follows that both b_k and b_m are placed below e and the endpoint of b_k at H_ℓ is v or a vertex of H_ℓ to the left of v while the endpoint of b_m at H_ℓ is v or a vertex to the right of v. Let C be the convex chain that is a part of H_ℓ between the endpoints of b_k and b_m on H_ℓ. Suppose that C includes at least one edge. The bridge b_k has to form an angle not less than 180 degrees with the leftmost edge of C and symmetrically the bridge b_m has to form an angle not less than 180 degrees with the rightmost edge of C. However, this is impossible because the bridges b_k and b_m are below the edge e of the upper hull of S with endpoints on H_k and H_m so they form an angle less than 180 degrees. We conclude that C consists solely of v and consequently v is an endpoint of both b_k and b_m. See Figure 2. □

The following folklore lemma follows easily by a standard case analysis (cf. [5, 7, 8]). It implies that the recursive depth of the procedure *Bridge* is $O(\log n)$.

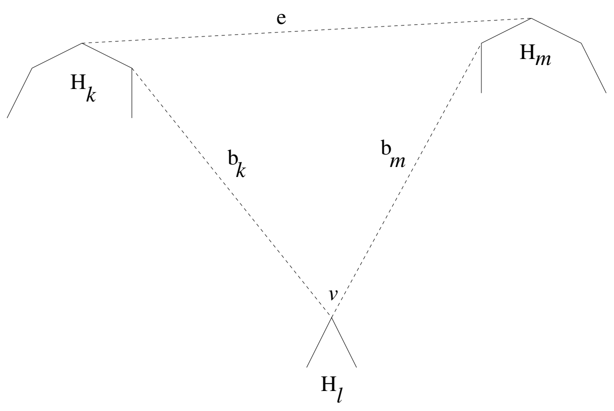


Figure 2: The final case in the proof of Lemma 2.

Lemma 3 *Let S_1, S_2 be two n-point sets in the Euclidean plane separated by a vertical line. Let H_1, H_2 be the upper hulls of S_1, S_2, respectively. Suppose that each of H_1 and H_2 has at least three vertices. Next, let m_1, m_2 be the median vertices of H_1, H_2, respectively. Suppose that the segment connecting m_1 with m_2 is not the bridge between H_1 and H_2. Then, the vertices on H_1 either to the left or to the right of m_1, or the vertices on H_2 either to the left or to the right of m_2 cannot be an endpoint of the bridge between H_1 or H_2.*

procedure $Bridge(H'_\ell, H'_m)$
Input: A continuous fragment H'_ℓ of the upper hull H_ℓ of the upper-hull subsequence assigned to a node ℓ and a continuous fragment H'_m of the upper hull H_m of the upper-hull subsequence assigned to the node m.
Output: The bridge between H'_ℓ and H'_m. Moreover, all points in the upper-hull subsequence held in the nodes ℓ and m placed under the bridge are marked as not qualifying for the convex hull of S.

1. If H'_ℓ or H'_m has at most two vertices then compute the bridge between H'_ℓ and H'_m by binary search. Next, mark all the points in the upper-hull subsequence between the endpoints of the found bridge that are assigned to the nodes ℓ or m as not qualifying for vertices of the upper hull of S and stop.

2. Find a median m_1 of H'_ℓ and a median m_2 of H'_m.

3. If the straight line passing through m_1 and m_2 is a supporting line for both H'_ℓ and H'_m then mark all the points in the upper-hull subsequence between m_1 and m_2 that are assigned to the nodes ℓ or m as not qualifying for vertices of the upper hull of S and stop.

4. Otherwise, call $Bridge(H''_\ell, H''_m)$, where either $H'_\ell = H''_\ell$ and H''_m is obtained from H'_m by removing vertices on the appropriate side of the median of H'_m or *vice versa*, according to Lemma 2.

The procedure $NewLowerHull(H'_\ell, H'_m)$ is defined analogously.

As in case of the procedure $QuickConvexHull(S)$, each step of $NewConvexHull(S)$, but for the calls to $NewUpperHull(S)$ and $NewLowerHull(S)$, can be done in $O(1)$ rounds on the congested clique by [6]. Furthermore, the first, next to the last, and last steps of $NewUpperHull(S)$ require $O(1)$ rounds. By Lemma 2, the recursion depth of the procedure $Bridge$ is logarithmic in n. The crucial observation is now that consequently the nodes ℓ and m need to exchange $O(\log n)$ messages in order to implement $Bridge(H_\ell, H_m)$. In particular, they need to inform each other about the current medians and in case H'_ℓ or H'_m contains at most two vertices, the node ℓ or m needs to inform about the situation and the two vertices the other node. In consequence, by Lemma 1, these two nodes can implement $Bridge(H_\ell, H_m)$ by sending a single message to each other in each round in a sequence of $O(\log n)$ consecutive rounds. It follows that all the n^2 calls of $Bridge(H_\ell, H_m)$ can be implemented in parallel in $O(\log n)$ rounds. Note that in each of the $O(\log n)$ rounds, each clique node sends at most one message to each other clique node, so in total, each node sends at most $n - 1$ messages to the other nodes in each of these rounds. It follows that $NewUpperHull(S)$ and symmetrically $NewLowerHull(S)$ can be implemented in $O(\log n)$ rounds on the congested clique. We conclude that $NewConvexHull(S)$ can be done in $O(\log n)$ rounds on the congested clique.

Theorem 4 *Consider a congested n-clique network, where each node holds a batch of n points in the Euclidean plane specified by $O(\log n)$-bit coordinates. The convex hull of the set S of the n^2 input points can be computed by the procedure $NewConvexHull(S)$ in $O(\log n)$ rounds on the congested clique.*

5 Point Set Triangulation in $O(\log^2 n)$ Rounds on Congested Clique

Our method of triangulating a set of n^2 points in the congested n-clique model initially resembles that of constructing the convex hull of the points. That is, first the input point set is sorted by x-coordinates. Then, each node triangulates its sorted batch of n points locally. Next, the triangulations are pairwise merged and extended to triangulations of doubled point sets by using the procedure $Merge$ in parallel in $O(\log n)$ phases. In the general case, the procedure $Merge$ calls the procedure $Triangulate$ in order to triangulate the area between the sides of the convex hulls of the two input triangulations, facing each other.

The main idea of the procedure $Triangulate$ is to pick a median vertex on the longer of the convex hulls sides and send its coordinates and the coordinates of its neighbors to the nodes holding the facing side of the other hull. The latter nodes send back candidates (if any) for a mate of the median vertex so that the segment between the median vertex and the mate can be an edge of a triangulation extending the input ones. The segment is used to split the area to triangulate into two that are triangulated by two recursive calls of $Triangulate$ in parallel. Before the recursive calls the edges of the two polygons surrounding the two areas are moved to new node destinations so each of the polygons is held by a sequence of consecutive clique nodes. This is done by a global routing in $O(1)$ rounds serving all parallel calls of $Triangulate$ on a given recursion level, for a given phase of $Merge$ (its first argument).

Since the recursion depth $Triangulate$ is $O(\log n)$ and $Merge$ is run in $O(\log n)$ phases, the total number of required rounds becomes $O(\log^2 n)$.

To simplify the presentation, we shall assume that the size n of the clique network is a power of 2.

procedure $Triangulation(S)$

1. Sort the points in S by their x-coordinates so each node receives a subsequence consisting of n consecutive points in S, in the sorted order.

2. Each node sends the first point and the last point in its subsequence to the other nodes.

3. Each node q constructs a triangulation $T_{q,q}$ of the points in its sorted subsequence locally.

4. For $1 \le p < q \le n$, $T_{p,q}$ will denote the already computed triangulation of the points in the sorted subsequence held in the nodes p through q. For $i = 0, \log n - 1$, in parallel, for $j = 1, 1 + 2^{i+1}, 1 + 22^{i+1}, 1 + 32^{i+1}, \ldots$ the union of the triangulations $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$ is transformed to a triangulation $T_{j,j+2^{i+1}-1}$ of the sorted subsequence held in the nodes j through $j + 2^{i+1} - 1$ by calling the procedure $Merge(i, j)$.

procedure $Merge(i, j)$
Input: A triangulation $T_{j,j+2^i-1}$ of the subsequence held in the nodes j through $j + 2^i - 1$ and a triangulation $T_{j+2^i,j+2^{i+1}-1}$ of the subsequence held in the nodes $j+2^i$ through $j + 2^{i+1} - 1$,.
Output: A triangulation $T_{j,j+2j+1-1}$ of the subsequence held in the nodes j through $j + 2^{j+1} - 1$.

1. Compute the bridges between the convex hulls of $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$. Determine the polygon P formed by the bridges between the convex hulls of $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$, the right side of the convex hull of $T_{j,j+2^i-1}$, and the left side of the convex hull of $T_{j+2^i,j+2^{i+1}-1}$ between the bridges.

2. $Triangulate(P, j, j + 2^{i+1} - 1)$

procedure $Triangulate(P, p, q)$
Input: A simple polygon P composed of two convex chains facing each other on opposite sides of a vertical line and two edges crossing the line, held in nodes p through q, with $p < q$.
Output: A triangulation of P held in nodes p through q.

1. If $p = q$ then the p node triangulates P locally and terminates the call of the procedure.

2. The nodes p through q determine the lengths of the convex chains on the border of P and the node holding the median vertex v of the longest chain (in case of ties, the left chain) sends the coordinates of v and the adjacent vertices on the chain to the other nodes p through q.

3. The nodes holding vertices of the convex chain that is opposite to the convex chain containing v determine if they hold vertices u that could be connected by a segment with v within P. They verify if the segment (v, u) is within the intersection of the union of the half-planes induced by the edges adjacent to v on the side of P with the union of the half-planes induced by the edges adjacent to u on the side of P. If so, they send one such a candidate vertex u to the node holding v.

4. The node holding v selects one of the received candidate vertices u as the mate and sends its coordinates to the other nodes p through q.

5. The nodes p through q split the polygon P into two subpolygons P_1 and P_2 by the edge (v, u) and by exchanging messages in $O(1)$ rounds compute the new destinations for the edges of the polygons P_1 and P_2 so P_1 can be held in nodes p through r_1 and P_2 in the nodes r_2 through q, where $p \leq r_1 \leq r_2 \leq q$ and $r_1 = r_2$ or $r_2 = r_1 + 1$.

6. A synchronized global routing in $O(1)$ rounds corresponding to the current phase of the calls to the procedure $Merge$ (given by its first argument) and all parallel calls of the procedure $Triangulate$ on the same recursion level is implemented. In particular, the edges of P_1 and P_2 are moved to the new consecutive destinations among nodes p through q.

7. In parallel, $Triangulate(P_1, p, r_1)$ and $Triangulate(P_2, r_2, q)$ are performed.

At the beginning, we have outlined our triangulation method, in particular the procedures forming it, in a top-down fashion. We now complement this outline with a bottom-up analysis. All steps of the procedure $Triangulate(P, p, q)$ but for the recursive calls in the last step and the next to the last step can be implemented in $O(1)$ rounds, using only the nodes p through q. The next to the last step is a part of the global routing. It serves all calls of the procedure $Triangulate$ on the same recursion level for a given phase of the parallel calls of procedure $Merge(i, \ )$, i.e., for given i. Since each node is involved in at most two of the aforementioned calls of $Triangulate$ that cannot be handled locally, the global routing, implementing the next to the last step of $Triangulate$, requires $O(1)$ rounds. Since the recursion depth of $Triangulate$ is $O(\log n)$, $Triangulate$ takes $O(\log n)$ rounds. The first step of the procedure $Merge(i, j)$, i.e., constructing the bridges between the convex hulls, can be implemented in $O(\log n)$ rounds by using the convex hull algorithm from Section 4 on nodes i through $i + 2^{i+1} - 1$. The second step can easily be implemented in $O(1)$ rounds using the aforementioned nodes. Finally, the call to $Triangulate$ in the last step of $Merge$ requires $O(\log n)$ rounds by our analysis of this procedure. Again, it can be done by nodes j through $j + 2^{i+1} - 1$ but for the last steps of calls to $Triangulate$ that are served by the discussed, synchronized global routing in $O(1)$ rounds. We conclude that $Merge(i, j)$ can be implemented in $O(\log n)$ rounds. Finally, all steps in $Triangulation(S)$ except the one involving parallel calls to $Merge(i, j)$ in $O(\log n)$ phases can be done in $O(1)$ rounds. For a given phase, i.e., given i, each node is involved in $O(1)$ calls of $Merge(i, j)$ but for the next to the last steps in $Triangulate$ that for a given recursion level of $Triangulate$ are implemented by the join global routing in $O(1)$ rounds. It follows from our analysis of $Merge(i, j)$ and $i = O(\log n)$ that $Triangulate(S)$ can be implemented in $O(\log^2 n)$ rounds.

Theorem 5 *Consider a congested n-clique network, where each node holds a batch of n points in the Euclidean plane specified by $O(\log n)$-bit coordinates. A triangulation of the set S of the n^2 input points can be computed by the procedure $Triangulation(S)$ in $O(\log^2 n)$ rounds on the congested clique.*

6 Remarks

The primary difficulty in the design of efficient parallel algorithms for the Voronoi diagram of a planar point set using a divide-and-conquer approach is the efficient parallel merging of Voronoi diagrams [1, 11]. In the full version of this paper [4], we show that when the n^2 input points with $O(\log n)$-bit coordinates are drawn uniformly at random from a unit square then the expected number of rounds required to build their Voronoi diagram on the congested clique is $O(1)$.

Acknowledgments

This research was partially supported by Swedish Research Council grants 621-2017-03750 and 2018-04001, and JSPS KAKENHI JP20H05964.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel Computational Geometry. *Algorithmica*, 3: 293–327, 1988. Preliminary version in *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, FOCS 1985, pp. 468–477, 1985.

[2] S.G. Akl. Optimal parallel algorithms for computing convex hulls and for sorting. *Computing*, 33(1): 1–11, 1984.

[3] K. Censor-Hillel, P. Kaski, J.H. Korhonen, C. Lenzen, C., A. Paz, and J. Suomela. Algebraic Methods in the Congested Clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC 2015, pp. 143–152, 2015.

[4] J. Jansson, C. Levcopoulos, and A. Lingas. Convex Hulls and Triangulations of Planar Point Sets on the Congested Clique. arXiv:2305.09987, 2023.

[5] D.G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1): 287–299, 1986.

[6] C. Lenzen. Optimal Deterministic Routing and Sorting on the Congested Clique. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC 2013, pp. 42–50, 2013.

[7] M.H. Overmars and J. Van Leeuwen. Maintenance of Configurations in the Plane. *Journal of Computer and System Sciences*, 23(2): 166–204, 1981.

[8] F. Preparata. An optimal real-time algorithm for planar convex hulls. *Communications of the ACM*, 22(7): 402–405, 1979.

[9] J. Ramesh and S. Suresha. Convex Hull - Parallel and Distributed Algorithms. Technical Report, Stanford University, U.S.A., 2016.

[10] P. Robinson. What Can We Compute in a Single Round of the Congested Clique? arXiv:2210.02638, 2022.

[11] B. C. Vemuri, R. Varadarajan and N. Mayya. An Efficient Expected Time Parallel Algorithm for Voronoi Construction. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, SPAA 1992, pp. 392–401, 1992.

Lower Bounds for the Thickness and the Total Number of Edge Crossings of Euclidean Minimum Weight Laman Graphs and (2,2)-Tight Graphs

Yuki Kawakami[*] Shun Takahashi[†] Kazuhisa Seto[‡] Takashi Horiyama[§] Yuki Kobayashi[¶]
Yuya Higashikawa[||] Naoki Katoh[**]

Abstract

We explore the maximum total number of edge crossings and the geometric thickness of the Euclidean minimum-weight (k, ℓ)-tight graph on a planar point set P. In this paper, we show that $(10/7 - \epsilon)|P|$ and $(11/6 - \epsilon)|P|$ are lower bounds for the maximum total number of edge crossings for any $\epsilon > 0$ in cases $(k, \ell) = (2, 3)$ and $(2, 2)$, respectively. We also show that the lower bound for the geometric thickness is 3 for both cases. In the proofs, we apply the method of arranging isomorphic units regularly. While the method is developed for the proof in case $(k, \ell) = (2, 3)$, it also works for different ℓ.

1 Introduction

A bar-joint framework is one of the main frameworks studied in combinatorial rigidity theory. It consists of rigid straight-line bars and their joints, where joints have flexibility on the angles of their incident bars. We can discuss the bar-joint framework as a graph of combinatorial theory by a mapping each joint to a vertex and each bar to a straight-line edge [3]. One of the most fundamental results in combinatorial rigidity theory asserts that a graph G realized on a generic point set (i.e., the set of the coordinates is algebraically independent over the rational field) is rigid if and only if G contains a spanning Laman subgraph [6]. A graph $G = (V, E)$ is a *Laman graph* if it satisfies $|E| = 2|V| - 3$ and $|E(H)| \leq 2|V(H)| - 3$ for any subgraph H of G with $E(H) \neq \emptyset$. Laman graphs appear in a wide range of applications, not only statics but also mechanical design such as linkages, design of CAD systems, analysis of protein flexibility, and sensor network localization [9, 10].

The concept of the sparsity condition of a Laman graph is generalized to a *(k, ℓ)-tight graph* (see, e.g., [7]). The class of (k, ℓ)-tight graphs includes important graphs: a Laman graph is a $(2, 3)$-tight graph, and a spanning tree being studied in various fields is a $(1, 1)$-tight graph. Furthermore, the class of $(2, \ell)$-tight graphs, including Laman graphs, plays an important role in the 2-dimensional bar-joint framework. For example, a $(2, 2)$-tight graph realized on a generic point set, one of the graphs focused on in this paper, is minimally rigid when the joints are constrained to lie on the surface of a cylinder (since this surface allows two independent rigid-body motions) [8].

In this paper, we focus on the edge crossing of Laman graphs and $(2, 2)$-tight graphs. In order to realize a graph as a bar-joint framework on the plane in the real world, it is important to consider its edge-crossing. Thus, one of our concerns is the graphs that maximize the total number of edge crossings. Another concern is the graphs that maximize the geometric thickness. The geometric thickness of graph G is the smallest number of layers necessary to partition the edge set of G into layers so that no layers have edge crossing (see, e.g., [4]).

Thus, more specifically, we at first focus on the total number of edge crossings and the geometric thickness of the Euclidean minimum-weight Laman graphs. The *Euclidean minimum-weight Laman graph* on point set P, denoted by MLG(P), is the Laman graph with the minimum total edge length among all Laman graphs on P. Then, we also focus on those of the Euclidean minimum-weight $(2, 2)$-tight graphs, where the *Euclidean minimum-weight (k, ℓ)-tight graph* on P, denoted by (k, ℓ)-MTG(P), is defined similarly as a generalization of MLG(P). While the Euclidean minimum-weight spanning tree on P is always planar (i.e., it has no edge crossings), MLG(P) may have some edge crossings (see e.g., Fig. 2).

Bereg et al. [1] showed many properties of MLG(P), e.g., 6-planarity, non-crossing of three edges, and the

[*]Graduate School of Information Science and Technology, Hokkaido University, kawakami.yuki.k4@elms.hokudai.ac.jp
[†]Graduate School of Information Science and Technology, Hokkaido University, shushun-bb14@eis.hokudai.ac.jp
[‡]Faculty of Information Science and Technology, Hokkaido University, seto@ist.hokudai.ac.jp
[§]Faculty of Information Science and Technology, Hokkaido University, horiyama@ist.hokudai.ac.jp
[¶]Department of Engineering, Osaka Metropolitan University, kobayashi@osaka-cu.ac.jp
[||]Graduate School of Information Science, University of Hyogo, higashikawa@sis.u-hyogo.ac.jp
[**]Graduate School of Information Science, University of Hyogo, naoki.katoh@gmail.com

implication $\mathrm{MLG}(P) \subseteq 1\text{-}\mathrm{GG}(P)$ on the edges of the graphs, where $1\text{-}\mathrm{GG}(P)$ is a 1-Gabriel graph. From the 6-planarity of $\mathrm{MLG}(P)$, they showed that the upper bound for the total number of edge crossings of $\mathrm{MLG}(P)$ is $6|P| - 9$. They also showed that the lower bound for the total number of edge crossings of $\mathrm{MLG}(P)$ is $|P| - 3$. Later, Kobayashi et al. [5] improved the upper and lower bounds for the total number of edge crossings of $\mathrm{MLG}(P)$ to $2.5|P| - 5$ and $(1.25 - \epsilon)|P|$ for any $\epsilon > 0$, respectively. Unfortunately, a gap between those bounds still exists. As for the geometric thickness of $\mathrm{MLG}(P)$, since the geometric thickness of $1\text{-}\mathrm{GG}(P)$ is at most 4 [2], $\mathrm{MLG}(P) \subseteq 1\text{-}\mathrm{GG}(P)$ in [1] implies that the upper bound for the geometric thickness of $\mathrm{MLG}(P)$ is 4. On the other hand, its lower bound is 2 since $\mathrm{MLG}(P)$ may have some edge crossings. Thus, we also have a gap in the geometric thickness.

Furthermore, the total number of edge crossings and the geometric thickness of (k, ℓ)-$\mathrm{MTG}(P)$ for general k and ℓ is in our interest. Bereg et al. [1] showed that (k, ℓ)-$\mathrm{MTG}(P)$ is $(6k^2 + 4k - 10)$-planar. In other words, each edge of (k, ℓ)-$\mathrm{MTG}(P)$ cross at most $(6k^2 + 4k - 10)$ other edges. According to this result, it is easy to see that the total number of edge crossings of (k, ℓ)-$\mathrm{MTG}(P)$ is at most $(6k^2 + 4k - 10)(k|P| - \ell)/2$. As Bereg et al. told in [1], this upper bound is not tight, as we can see the bound is $22|P| - 33$ in case $k = 2$ and $\ell = 3$. There are many open questions regarding the total number of edge crossings and the geometric thickness of the (k, ℓ)-$\mathrm{MTG}(P)$. As a first step for general k and ℓ, we focus on $(2, 3)$-$\mathrm{MTG}(P)$ and $(2, 2)$-$\mathrm{MTG}(P)$.

Our contribution: At first, we improve the lower bound for the total number of edge crossings of $\mathrm{MLG}(P)$. Our idea for the proof is based on the method by Kobayashi et al. [5]: arrange the same units on a circumference, where each unit consists of carefully positioned five points. We extend this method by alternately arranging two types of units on a circumference. Each of both units consists of eight points, and their arrangement is well determined so as to derive isomorphic MLGs and not to interfere with each other. By alternately arranging these two types of units, we derive the lower bound $(1.42 - \epsilon)|P|$ for any $\epsilon > 0$.

Our extended method has the possibility to derive a lower bound for general cases. To show the power of our method, we apply it to different ℓ. More precisely, We derive a lower bound for the total number of edge crossings of $(2, 2)$-$\mathrm{MTG}(P)$ by regularly arranging different units made under the same design: Each unit differs in only one parameter regarding width, while all other parameters are the same for all units. By regularly arranging these units, we derive the lower bound $(1.83 - \epsilon)|P|$ for any $\epsilon > 0$, while no lower bounds were known. Our results on $(2, 3)$-$\mathrm{MTG}(P)$ and $(2, 2)$-$\mathrm{MTG}(P)$ suggest that the total number of edge crossings depends on parameter ℓ. Recall that the upper bound by Bereg et al. [1] was with parameter k. Thus, we can open the door for the discussion with general k and ℓ.

We also address the lower bounds for the geometric thickness of $\mathrm{MLG}(P)$ and $(2, 2)$-$\mathrm{MTG}(P)$. We use the edge-crossing graph (also called crossing dual graph) of a graph G. Each vertex and edge of the edge-crossing graph corresponds to an edge of G and the edge crossing of two edges of G, respectively. Interestingly, the chromatic number of the edge-crossing graph is equal to the geometric thickness of the original graph G. We show an instance of the edge-crossing graph of $\mathrm{MLG}(P)$ that contains a cycle of odd length. Since this implies its chromatic number is at least 3, we can improve the lower bound to 3. In a similar way, we can also derive the same lower bound for the geometric thickness of $(2, 2)$-$\mathrm{MTG}(P)$.

2 Preliminaries

2.1 Minimum-weight (k,ℓ)-tight graphs

A graph $G = (V, E)$ is a *(k, ℓ)-sparse graph* $(0 \leq \ell \leq 2k - 1)$ if it satisfies $|E(H)| \leq k|V(H)| - \ell$ for any subgraph H of G with $E(H) \neq \emptyset$, where $E(H)$ denotes the set of edges of H. A (k, ℓ)-sparse graph is called a *(k, ℓ)-tight graph* if it has exactly $k|V(G)| - \ell$ edges. In particular, $(1, 1)$-tight graph, i.e., the graph for the case $k = \ell = 1$, is called a spanning tree, and $(2, 3)$-tight graph is called a Laman graph.

A *geometric graph* $G(P) = (P, E_{\mathbf{p}})$ is obtained by embedding a graph $G = (V, E)$ into a 2-dimensional Euclidean plane by a bijection $\mathbf{p} : V \to P$. Each vertex $v_i \in V$ of graph G is mapped to a point $\mathbf{p}(v_i) = p_i$, and each edge $(v_i, v_j) \in E$ is mapped to a line segment $\mathbf{p}(v_i)\mathbf{p}(v_j) \in E_{\mathbf{p}}$. In this paper, we denote p_ip_j as both the line segment $\mathbf{p}(v_i)\mathbf{p}(v_j)$ and the edge (v_i, v_j). The weight of edge p_ip_j is the Euclidean distance between two points p_i and p_j, denoted by $\|p_ip_j\|$.

The (k, ℓ)-tight graph with the minimum total edge weight among all (k, ℓ)-tight graphs on P is called the *Euclidean minimum-weight tight graph* on P, and denoted by (k, ℓ)-$\mathrm{MTG}(P)$. In case $k = 2$ and $\ell = 3$, (2,3)-MTG(P) is also called the *Euclidean minimum-weight Laman graph* on P, and denoted by $\mathrm{MLG}(P)$. Throughout the paper, we assume that no three points in P are collinear and that all distances between two points in P are distinct, called *semi-generic*.

The following lemma by Bereg et al. [1] is a good tool for distinguishing whether an edge is in the $\mathrm{MLG}(P)$.

Lemma 1 ([1] Lemma2.2) *Let P be a semi-generic point set in the plane, $Q \subseteq P$, and $a, b \in Q$. Also let E' be the set of line segments $\{pq \mid p, q \in Q, p \neq q, \|pq\| < \|ab\|\}$. If there exists a subset of E' that* *induces a Laman graph on Q, then $ab \notin E(MLG(P))$ holds.*

As in the following lemma, we can extend the above lemma to the general case, i.e., (k,ℓ)-MTG for general k and ℓ. We use it to prove the lower bounds for the total number of edge crossings of MLG(P) and $(2,2)$-MTG(P).

Lemma 2 *Let P be a semi-generic point set in the plane, $Q \subseteq P$, and $a, b \in Q$. Also let E' be the set of line segments $\{pq \mid p, q \in Q, p \neq q, \|pq\| < \|ab\|\}$. If there exists a subset of E' that induces a (k,ℓ)-tight graph on Q, then $ab \notin E((k,\ell)\text{-}MTG(P))$ holds.*

Sketch of the proof. Lemma 1 is the special case for Laman graphs ($k = 2, \ell = 3$). Bereg et al. [1] proved the lemma by utilizing the property of 2-dimensional rigid matroids. For general k and ℓ ($0 \leq \ell \leq 2k-1$), we can use (k,ℓ)-sparsity matroids [7], i.e., a generalization of 2-dimensional rigid matroids. □

2.2 Geometric thickness of geometric graphs

Our focus is the crossings of the edges in a geometric graph $G(P) = (P, E)$. Two edges e and e' ($\in E$) are *crossing* if and only if they have a common point other than their both ends. We denote the total number of edge crossings in $G(P)$ as $\sigma(G(P))$. The *geometric thickness* of $G(P)$ is defined as the minimum positive integer t satisfying the following conditions:

- $\cup_{i=1}^{t} E_i = E$.
- For any integer i ($1 \leq i \leq t$), geometric graph $G_i(P) = (P, E_i)$ is non-crossing (i.e., $G_i(P)$ is a plane graph).

Suppose that we are given a geometric graph $G(P)$ with geometric thickness t, and that we partition E into $t-1$ edge sets $E_1, E_2, \ldots, E_{t-1}$. Then, at least one $G_i(P) = (P, E_i)$ has edge crossing.

We introduce edge-crossing graphs of geometric graphs to understand the geometric thickness. Given a geometric graph $G(P) = (P, E)$, its edge-crossing graph(W, F) is defined as follows: each vertex $e \in W$ corresponds to edge $e \in E$, and edge (e, e') is in F if and only if edges e and e' cross each other in $G(P)$. The following relationship exists between the geometric thickness of a geometric graph and the chromatic number of the edge-crossing graph.

Lemma 3 ([5]) *The geometric thickness of a geometric graph $G(P)$ and the chromatic number of the edge-crossing graphof $G(P)$ are equal.*

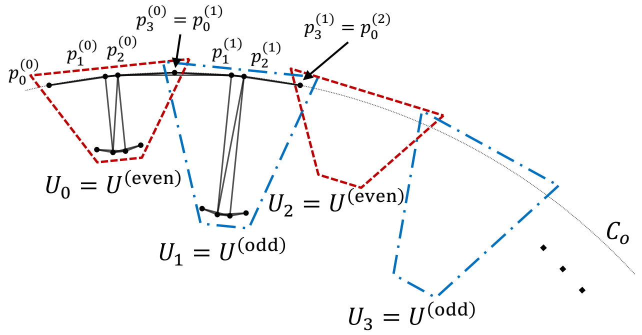


Figure 1: Arranging alternately $U^{(\text{even})}$ and $U^{(\text{odd})}$

3 Lower bounds

In this section, we show the lower bounds for the geometric thickness and the total number of edge crossings of MLG(P) and (k,ℓ)-MTG(P). To improve the lower bound for the total number of edge crossings, we use units consisting of several points. For MLG(P), the lower bound is derived by counting the total number of edge crossings of MLG on a point set with alternately arranged two types of units. For $(2,2)$-MTG(P), we consider a point set regularly arranging different units made under the same design. We also improve the lower bound for the geometric thickness by showing that the geometric thickness of both MLG(P) and $(2,2)$-MTG(P) is 3. We focus on MLG(P) in section 3.1, and $(2,2)$-MTG(P) in section 3.2.

3.1 MLG(P)

Kobayashi et al. [5] derived the lower bound by regularly arranging one type of unit consisting of five points. We improve the lower bound by extending the idea to arrange two types of units $U^{(\text{even})}$ and $U^{(\text{odd})}$ alternately, where each unit consists of eight points. The alternate arrange of $U^{(\text{even})}$ and $U^{(\text{odd})}$ is illustrated in Figure 1. Both of these two units $U^{(\text{even})}$ and $U^{(\text{odd})}$ are positioned so as to derive the isomorphic MLGs in all units. The alternation of two different types of units instead of the same type will give two crossings between the neighboring units. Let t denote the number of units we arrange, and $P(t)$ denote the point set when t units are arranged. Also, we denote the i-th unit as U_i ($0 \leq i \leq t-1$) and the point p_x in unit U_i as point $p_x^{(i)}$.

First, we describe the details of units $U^{(\text{even})}$ and $U^{(\text{odd})}$, and show the MLG on the point set of a unit $U^{(\text{even})}$. Unit $U^{(\text{even})}$ is obtained by translating and rotating the eight points in Figure 2, where six parameters $d, \delta, \delta', \tau, \tau'$ and h_e are positive real numbers. Unit $U^{(\text{odd})}$ is obtained by replacing the parameter h_e in $U^{(\text{even})}$ with $h_o = h_e + d + 2\tau$. We carefully determine the parameters so that the two MLGs on point

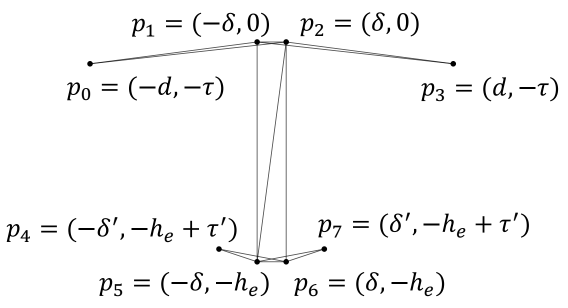


Figure 2: MLG($U^{(\text{even})}$)

sets $U^{(\text{even})}$ and $U^{(\text{odd})}$ become isomorphic.

Lemma 4 *Suppose that the parameters $d, \delta, \delta', \tau, \tau'$ and h_e or h_o satisfy the following conditions:*

(A) $\delta' > 3\delta, \delta + \tau'$

(B) $d > \delta + \tau, 2\delta'$

(C) $d + \delta + 2\tau + \tau' < h_e, h_o < \frac{\delta(\delta' - 3\delta)}{2\tau'}, \frac{(d-\delta')^2 - (\delta' - \delta)^2}{2\tau}$

Then, the MLG on a point set $U^{(even)}$ is the geometric graph illustrated in Figure 2. The MLG on $U^{(odd)}$ is the graph obtained by replacing h_e in Figure 2 with h_o.

Sketch of the proof. We prove this lemma by showing that the geometric graph illustrated in Figure 2 is a Laman graph and that all edges not illustrated in Figure 2 are not included in MLG($U^{(\text{even})}$). For discriminating whether an edge is in MLG($U^{(\text{even})}$) or not, we use Lemma 1. For example, let us focus edge p_0p_3. Suppose that we have a point set $Q = \{p_0, p_1, p_2, p_3\}$. Then, we can obtain the set $E_{p_0p_3}$ of line segments shorter than $\|p_0p_3\|$ as $E_{p_0p_3} = \{p_0p_1, p_0p_2, p_1p_2, p_1p_3, p_2p_3\}$. Since graph $(Q, E_{p_0p_3})$ is a Laman graph on Q, $p_0p_3 \notin$ MLG($U^{(\text{even})}$) holds by Lemma 1. By a similar argument, we check all edges not illustrated in Figure 2 one by one. As a result, we can prove the first statement on $U^{(\text{even})}$. From the argument for constructing $U^{(\text{odd})}$, we can say that same holds for MLG($U^{(\text{odd})}$). □

Next, we consider the point set $P(t)$ obtained by arranging t units. In case $t = 1$, $P(1)$ is $U^{(\text{even})}$ itself, and thus the graph in Figure 2 is MLG($P(1)$). In case $t > 1$, we alternately arrange $U^{(\text{even})}$ and $U^{(\text{odd})}$ as in Figure 1. More precisely, for integer i $(0 \le i \le t-1)$, U_i is $U^{(\text{even})}$ if i is even and $U^{(\text{odd})}$ if i is odd. We arrange t units $U_0, U_1, \ldots, U_{t-1}$ as $P(t)$ so that every four points $p_0^{(i)}, p_1^{(i)}, p_2^{(i)}, p_3^{(i)}$ of unit U_i lie on the same circumference C_0. In addition, for all integer i $(0 \le i \le t-2)$ two points $p_3^{(i)}$ of unit U_i and $p_0^{(i+1)}$ of unit U_{i+1} are adjusted to the same position and are regarded as the same point. The following lemma tells which line segment is in the MLG on $P(t)$.

Lemma 5 *The set of edges in MLG($P(t)$) is a union of the set of edges in MLG(U_i) for $0 \le i \le t-1$ and the set of edges $p_2^{(i)}p_1^{(i+1)}$ between two neighboring units U_i and U_{i+1} for $0 \le i \le t-2$.*

Sketch of the proof. We can prove this lemma by the similar argument in Lemma 4. All edges not included in the edge set of the MLG on the point set arranged one of each unit are excluded by the same argument. For edges between different units, only edges $p_2^{(j)}p_1^{(j+1)}$ are included in the E(MLG($P(t)$)) and no other edges are included. For all integer j $(0 \le j \le t-1)$, the weight of edge $\|p_2^{(j)}p_1^{(j+1)}\|$ can be approximated to $2d$ by adjusting four parameters $\delta, \delta', \tau, \tau'$ very small while satisfying the conditions. For all even x, the weights of the edges $p_7^{(x)}p_4^{(x+1)}$ and $p_7^{(x)}p_1^{(x+1)}$ can be approximated to $\sqrt{5}d$ by making h_e a small value while satisfying the condition (C)(and symmetrically for edges $p_4^{(x)}p_7^{(x-1)}$ and $p_4^{(x)}p_3^{(x-1)}$). The weights of the other edges between the different units are larger than those of these edges. Note that all edges included in the edges of the MLG on the point set arranged one of each unit U_i are shorter than either edge $p_7^{(i)}p_4^{(x+1)}$ or $p_7^{(x)}p_1^{(x+1)}$. If we consider an edge set E that is shorter than the weight $\min(\|p_7^{(x)}p_4^{(x+1)}\|, \|p_7^{(x)}p_1^{(x+1)}\|)$, then it contains a Laman graph on the point set $P(t)$. Therefore, the edges between the different units do not contain in E(MLG($P(t)$)) anything other than edge $p_2^{(i)}p_1^{(i+1)}$ from Lemma 1. □

Now let us count the number of edge crossings in the MLG on $P(t)$. For each unit U_i $(0 \le i \le t-1)$, we have eight crossings in MLG(U_i). In addition, for each neighboring units U_i and U_{i+1} $(0 \le i \le t-2)$, we have two crossings $(p_2^{(i)}p_1^{(i+1)}, p_1^{(i)}p_3^{(i)})$ and $(p_2^{(i)}p_1^{(i+1)}, p_0^{(i+1)}p_2^{(i+1)})$. Thus, the total number of edge crossings of MLG($P(t)$) is $8t + 2(t-1) = 10t - 2$. And since the number of points is $7t + 1$ in this case, we obtain the following equation:

$$\frac{\sigma(\text{MLG}(P(t)))}{|P(t)|} = \frac{10t - 2}{7t + 1} = \frac{10}{7} - \frac{24}{49t + 7}.$$

Here, we determin the radius R of circle C_0 as

$$R = \frac{\sqrt{((d-\delta)^2 + \tau^2)((d+\delta)^2 + \tau^2)}}{2\tau}.$$

By adjusting parameters $d, \delta, \delta', \tau, \tau', h_e$ and h_o so that they are satisfying conditions (A) to (C), $h_o = h_e + d + 2\tau$ and $2\pi R \gg 2t(d+\delta)$, the number of units t can be made arbitrarily large. In other words, for any $\epsilon > 0$, we can obtain a point set P whose MLG has at least $(\frac{10}{7} - \epsilon)|P|$ crossings. Although the above point set P is not semi-generic, we can obtain a set of semi-generic points by moving each point in P infinitesimally.

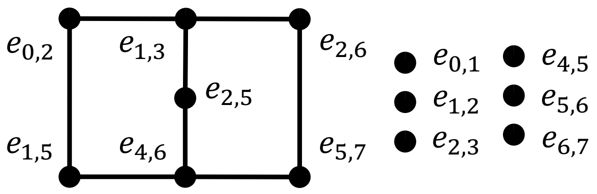


Figure 3: edge-crossing graphof MLG($U^{(\text{even})}$)

Theorem 6 *For any $\epsilon > 0$, there exists a set of semi-generic points P such that the total number of edge crossings of MLG(P) is greater than $(\frac{10}{7} - \epsilon)|P|$.*

Now, we focus on the geometric thickness of MLG(P). The graph shown in Figure 3 is the edge-crossing graphof MLG($U^{(\text{even})}$). Point p_ip_j in Figure 3 corresponds to edge p_ip_j in MLG($U^{(\text{even})}$), and edge $(p_{i_1}p_{j_1}, p_{i_2}p_{j_2})$ corresponds to a crossing of edges $p_{i_1}p_{j_1}$ and $p_{i_2}p_{j_2}$ in MLG($U^{(\text{even})}$). This graph contains a cycle of length 5. Hence, this graph is not 2-colorable. In other words, its chromatic number is 3 or more. Since the geometric thickness of a geometric graph is equal to the chromatic number of its edge-crossing graphfrom Lemma 3, the geometric thickness of MLG of $U^{(\text{even})}$ is 3 or more. Thus, we have the following theorem.

Theorem 7 *There exists a set of semi-generic points P such that the geometric thickness of MLG(P) is greater than or equal to 3.*

3.2 $(2, 2)$-MTG

In section 3.1, we alternately arranged two types of units $U^{(\text{even})}$ and $U^{(\text{odd})}$. In this subsection, we derive the lower bound for the total number of edge crossings of $(2, 2)$-MTG by a new approach: While we arrange mutually different t units, the position of the points in the units is designed so that the $(2, 2)$-MTGs on all units are isomorphic. As in section 3.1, we first describe each unit U_i and the rules for arranging t units. Let $P(t)$ denote the point set with t units. Then we discuss discuss the $(2, 2)$-MTG on the point set with t units, and finally the total number of edge crossings of the $(2, 2)$-MTG(P).

For each i in $0 \leq i \leq t-1$, unit U_i consists of six points. The relative position of the points in each unit is illustrated in Figure 4, and is determined by three parameters δ, ϵ and d_i. Two parameters δ, ϵ are common for all units, and parameter d_i is different in each unit. In each U_i, three points $p_0^{(i)}$, $p_3^{(i)}$ and $p_4^{(i)}$ (respectively, $p_1^{(i)}$, $p_2^{(i)}$ and $p_5^{(i)}$) have the same x-coordinate. The height of U_i is always ϵ. On the other hand, as i increases, the width of U_i also increases. As illustrated in Figure 5, we vertically arrange t units so that two points $p_3^{(i)}$ and $p_1^{(i+1)}$ (respectively, $p_4^{(i)}$ and $p_2^{(i+1)}$) in each of the neighboring units U_i, U_{i+1} have the same y-coordinate. The lengths $\|p_3^{(i)}p_1^{(i+1)}\|$ and $\|p_4^{(i)}p_2^{(i+1)}\|$

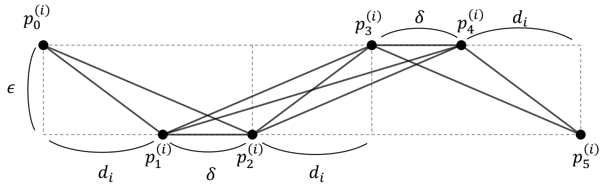


Figure 4: $(2, 2)$-MTG(U_i)

is fixed to h for all i's. Suppose that parameters d_i, δ, ϵ and h satisfy the following conditions:

(i) $\delta < d_0$

(ii) $d_{i+1} > 2d_i + \delta$

(iii) $\epsilon < \frac{(d_1 - d_0 - \delta)^2}{4h}, \frac{(d_1 - 2d_0 - \delta)}{2h}$

(iv) $h > 2d_t + \delta$

Each unit U_i $(1 \leq i \leq t-2)$ has a point set $Q = \{p_0^{(i)}, p_1^{(i)}, \ldots, p_5^{(i)}\}$. Let us focus on edge $p_0^{(i)}p_3^{(i)}$. Then, we can obtain the set E of edges whose length is shorter than $\|p_0^{(i)}p_3^{(i)}\| = 2d_i + \delta$. Actually speaking, E is the edges illustrated in Figure 4, and we can show that E is the set of edges of $(2, 2)$-MTG. Therefore, we can conclude that edge $p_0^{(i)}p_3^{(i)}$ is not included in $(2, 2)$-MTG(U_i) from Lemma 2. By the similar argument, we can show that all edges longer than $2d_i + \delta$ are not included in $(2, 2)$-MTG(U_i).

By a similar argument with Lemma 5, the set of edges in $(2, 2)$-MTG($P(t)$) is a union of the set of edges in $(2, 2)$-MTG(U_i) for $\leq i \leq t-1$ and the set of edges $p_3^{(i)}p_1^{(i+1)}$ and $p_4^{(i)}p_2^{(i+1)}$ between two neighboring units U_i and U_{i+1} for $0 \leq i \leq t-2$.

Since there are 5 crossings in each unit $U_i (0 \leq i \leq t-1)$ and 8 crossings for each neighboring U_j and U_{j+1} $(0 \leq j \leq t-2)$, the total number of edge crossings of $(2, 2)$-MTG is $5 \cdot t + 6(t-1) = 11t - 6$ when arranging t units. Since the number of points is $6t$ in this case, we obtain the following equation:

$$\frac{\sigma(\text{MLG}(P(t)))}{|P(t)|} = \frac{11t - 6}{6t} = \frac{11}{6} - \frac{1}{t}.$$

Thus, with a sufficiently large t, for any $\epsilon > 0$, there exists a point set P such that the total number of edge crossings is at least $(\frac{11}{6} - \epsilon)|P|$. As in the previous subsection, moving each point in P infinitesimally, we can obtain a set of semi-generic points.

Theorem 8 *For any $\epsilon > 0$, there exists a set of semi-generic points P such that the total number of edge crossings of MLG(P) is greater than $(\frac{11}{6} - \epsilon)|P|$.*

In the following, we discuss the thickness. Focus on the 5 edges $p_2^{(0)}p_4^{(0)}, p_3^{(0)}p_1^{(1)}, p_2^{(0)}p_5^{(0)}, p_4^{(0)}p_2^{(1)}, p_3^{(0)}p_5^{(0)}$ and their 5 edge crossings in $(2, 2)$-MTG($P(t)$) shown

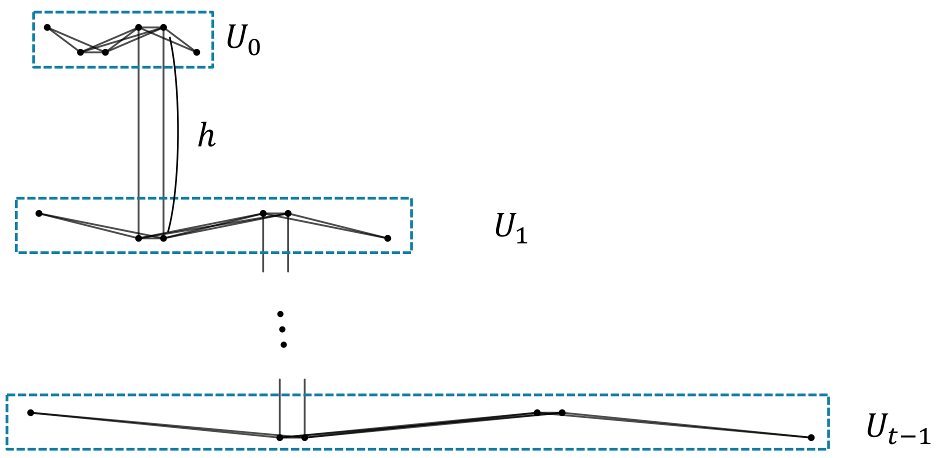


Figure 5: (2, 2)-MTG on a point set arranging t units

in Figure 5. Then we consider the edge-crossing graph for that part; there exists a cycle of length 5. Therefore, for the same reason as MLG(P) and Lemma 3, the geometric thickness of (2, 2)-MTG($P(t)$) is at least 3. Thus, we have the following theorem.

Theorem 9 *There exists a set of semi-generic points P such that the geometric thickness of* (2, 2)*-MTG*(P) *is greater than or equal to 3.*

4 Concluding remarks

As for the the total number of edge crossings, with the idea of regularly arranging different units, we improved the lower bound for MLG(P) and newly derived the lower bound for (2, 2)-MTG(P). As for the geometric thickness, we showed that the lower bounds for MLG(P) and (2, 2)-MTG(P) are 3 since we have a cycle of length 5 in each of their edge-crossing graphs.

A gap, however, still exists between the upper and lower bounds for the total number of edge crossings of MLG(P). One of the challenges is to fill this gap. There is also a large gap for (2, 2)-MTG(P). The reason for this large gap is due to the difficulty that the technique for the upper bound of MLG(P) cannot be directly applied to (2, 2)-MTG(P) since it may contain cliques of size 4. New techniques are necessary to address the upper bound for (2, 2)-MTG(P) (and also for general k and ℓ). For the lower bound, we believe that our technique of arranging different units made under the same design is promising for general k and ℓ.

A gap also exists for the geometric thickness of MLG(P). Kobayashi et al. [5] gave a suggestion for improving the upper bound. A planar triangle-free graph is 3-colorable, and the edge-crossing graph of MLG(P) is triangle-free. Thus, if we prove the planarity of the edge-crossing graph, the upper bound becomes 3 (i.e., we have the matching upper and lower bounds). As for the geometric thickness for (2, 2)-MTG(P), the upper bound is open. Moreover, it is not known whether the edge-crossing graph of (2, 2)-MTG(P) is triangle-free or not.

References

[1] S. Bereg, S.-H. Hong, N. Katoh, S.-H. Poon, and S. Tanigawa. On the edge crossing properties of euclidean minimum weight laman graphs. *Comput. Geom.*, 51:15–24, 2016.

[2] P. Bose, S. Collette, F. Hurtado, M. Korman, S. Langerman, V. Sacristán, and M. Saumell. Some properties of k-delaunay and k-gabriel graphs. *Comput. Geom.*, 46(2):131–139, 2013.

[3] J. E. Graver, B. Servatius, and H. Servatius. *Combinatorial rigidity*, pages 12–65. Number 2. American Mathematical Soc., 1993.

[4] P. C. Kainen. Thickness and coarseness of graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 39(1):88–95, 09 1973.

[5] Y. Kobayashi, Y. Higashikawa, and N. Katoh. Improving upper and lower bounds for the total number of edge crossings of euclidean minimum weight laman graphs. In *Proc. COCOON, LNCS 13025*, pages 244–256. Springer, 2021.

[6] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4(4):331–340, 1970.

[7] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 308(8):1425–1437, 2008.

[8] A. Nixon, J. C. Owen, and S. C. Power. Rigidity of frameworks supported on surfaces. *SIAM Journal on Discrete Mathematics*, 26(4):1733–1757, 2012.

[9] B. Servatius. The geometry of frameworks: Rigidity, mechanisms and cad. *MAA Notes*, pages 81–87, 2000.

[10] M. F. Thorpe and P. M. Duxbury. *Rigidity theory and applications*. Springer Science & Business Media, 1999.

Geometric Graphs with Unbounded Flip-Width*

David Eppstein† Rose McCarty‡

Abstract

We consider the flip-width of geometric graphs, a notion of graph width recently introduced by Toruńczyk. We prove that many different types of geometric graphs have unbounded flip-width. These include interval graphs, permutation graphs, circle graphs, intersection graphs of axis-aligned line segments or axis-aligned unit squares, unit distance graphs, unit disk graphs, visibility graphs of simple polygons, β-skeletons, 4-polytopes, rectangle of influence graphs, and 3d Delaunay triangulations.

1 Introduction

Flip-width is a new and very general notion of width in graphs, defined by Szymon Toruńczyk [28] using a cops-and-robbers game on graphs, and intended to capture graph structure in a way that allows for efficient parameterized algorithms. It is hoped that testing whether a given graph models a first-order formula in the logic of graphs can be solved efficiently when parameterized by flip-width and formula size, although currently this is known only for more limited classes of graphs [10].

Beyond potential algorithms, another purpose of flip-width is to unify incompatible notions of graph width, including bounded expansion and of twin-width. A graph family has *bounded expansion* if all shallow minors of its graphs are sparse [25]. It has bounded *twin-width* if its graphs can be reduced to one vertex by contracting pairs of vertices so that the subgraph of pairs of contracted vertices with inconsistent adjacencies maintains bounded degree throughout the contraction process [6]. The sparse graph families of bounded flip-width are exactly the families of bounded expansion, and every graph family of bounded twin-width has bounded flip-width [28]. It is easy to construct graph families that have bounded flip-width but neither bounded expansion nor bounded twin-width, such as the family of the graphs that are either subcubic or cographs. The subcubic graphs have bounded expansion but unbounded twin-width [3] while cographs reverse these inclusions. This union is not very natural, though; subcubic graphs and cographs have little in common. Can we find a natural family of graphs with bounded flip-width, but neither bounded twin-width nor bounded expansion?

Natural candidates include geometric graphs, whose vertices come from points or other simple objects in a geometric space, and whose edges are defined by simple geometric relations between these objects. However, planar graphs, and bounded-ply disk intersection graphs in bounded dimensions have bounded expansion [12, 24], as do sparse intersection graphs of connected subsets of a surface [12, 21]. To find the examples we seek, we need non-sparse graphs. Geometric graph theory contains many examples of highly structured but non-sparse graph families. Do any have bounded flip-width?

In this work we provide a negative answer for many standard geometric graphs. We find a class of subgraphs common to these graphs, which we call "interchanges" and which provide a winning strategy for a robber in the cops-and-robbers game used to define flip-width. A graph family that includes arbitrarily large interchanges has unbounded flip-width. Using this idea we show that interval graphs, permutation graphs, circle graphs, intersection graphs of simply-intersecting axis-aligned line segments, intersection graphs of axis-aligned unit squares, unit distance graphs, unit disk graphs, visibility graphs of simple polygons, β-skeletons, rectangle of influence graphs, and the graphs of 4-polytopes all have unbounded flip-width. We provide a different construction showing that the graphs of 3-dimensional Delaunay triangulations have unbounded flip-width.

For many of these graphs we prove more strongly that the radius-1 flip-width is unbounded and that these graphs are monadically independent, a related concept in the logic of graphs. A similar approach was used by Hliněný, Pokrývka, and Roy [17] to prove hardness of first-order model checking on graph classes with a specific type of interchange, which they call the "consecutive neighbourhood representation property". Other hardness results, as well as some efficient algorithms, have been obtained for various geometric graphs in [2, 14, 16, 17]. The proofs of such hardness results typically imply that the flip-width is unbounded, using the following key fact; a transduction of a class of bounded flip-width also has bounded flip-width [28]. Beyond extending these results to more graph classes, our approach has the advantages of only using first concepts, and of providing a concrete robber strategy and a specific bound on the radius.

*Research initiated at the 10th Workshop on Geometry and Graphs, Feb. 3–10, 2023, Bellairs Research Institute, Barbados. We thank the workshop participants (especially Jit Bose) and Szymon Toruńczyk for helpful conversations on this work.

†Department of Computer Science, University of California, Irvine. Research supported in part by NSF grant CCF-2212129.

‡Department of Mathematics, Princeton University. Research supported by the NSF under grant DMS-2202961.

2 Cops and robbers

Like treewidth [27] and bounded expansion [28], flip-width can be defined using a certain cops-and-robbers game. The games for treewidth and expansion involve "cops with helicopters", chasing a robber on a graph. The cops can occupy a limited number of graph vertices (initially, none); the robber can choose any starting vertex. In each time step, the cops announce where they will move next, the robber moves to escape them on a path through currently-unoccupied vertices, and then the cops fly directly to their new locations. The cops win by landing on the robber's current vertex, and the robber wins by evading the cops indefinitely. The treewidth of a graph is the maximum number of cops that a robber can evade, moving arbitrarily far on each move [27]. A family of graphs has bounded expansion if and only if, for some function f, a robber who moves $\leq r$ steps per move can be caught by $f(r)$ cops [28].

The same game can be described differently. Instead of occupying a vertex, the cops set up roadblocks on all edges incident to it. On each move, the cops announce which vertices will be blockaded next. Then, the robber moves along un-blockaded edges. Finally, the cops remove their current blockades and put up new blockades at the announced locations. The cops win by leaving the robber at an isolated vertex, unable to move. Flip-width is defined in the same way, but with more powerful cops. Instead of blockading a single vertex, they may "flip" any subset of vertices. This complements the subgraph induced by that subset: pairs of adjacent vertices become non-adjacent, and vice versa. Blockading a single vertex, for instance, takes two flips: one flip of the vertex and its neighbors, and one of just the neighbors. The first flip disconnects the given vertex, and the second restores its neighbors' adjacencies. It doesn't matter in which order these two flips (or any set of flips) is performed.

In the flipping game used to define flip-width, at any move, the cops may perform a limited number of flips, initially none. The robber chooses an arbitrary starting vertex. In each move, the cops announce their next set of flips. The robber moves on a path in the current flipped graph, to evade these flips. Then, the cops undo their current flips and perform the flips that they announced. The cops win by leaving the robber at an isolated vertex, unable to move, and the robber wins by avoiding this fate indefinitely. A family of graphs has *bounded flip-width* if, for some function f, $f(r)$ flips per move suffice to catch a robber who moves $\leq r$ steps per move. Similarly, the *radius-r flip-width* of a graph is the least number of flips required to catch a robber who moves $\leq r$ steps. Bounded flip-width implies bounded radius-r flip-width, but not vice versa; for instance, subdivisions of complete graphs have bounded radius-1 flip-width but unbounded flip-width. Conversely, unbounded radius-r flip-width implies unbounded flip-width, but not vice versa.

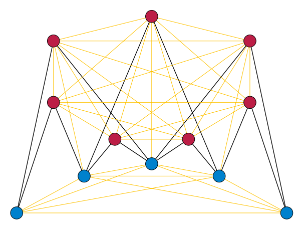


Figure 1: An interchange of order five, with lanes in blue and ramps in red. The yellow edges are optional.

Because flipping can simulate blockading, graphs of bounded treewidth also have bounded flip-width. However, graphs of unbounded treewidth may have bounded flip-width. For instance, all planar graphs have bounded flip-width but the planar graphs have unbounded treewidth.

3 Escaping through interchanges

In the treewidth game, escape strategies for the robber are modeled graph-theoretically by havens, certain functions from subsets of vertices to connected components of the subgraph formed by their removal [27]. In the same spirit, and using terminology following a road network metaphor, we define *interchanges*, structures in a graph which can be used to define an escape strategy for the robber in the flipping game.

Definition 1. An interchange of order n consists of:

- A linear sequence of n designated vertices, which we call *lanes*.
- More designated vertices, called *ramps*. Each ramp is associated with two lanes, and each two lanes that are $\leq n-3$ steps apart in the sequence have a ramp. (We do not require ramps for farther-apart lanes because they would not be of use to the robber.)
- An edge between each ramp and its two lanes.
- Optional edges between any two lanes or between any two ramps. These will be unused by the robber. Making them optional, rather than specifying their presence or absence, allows us to construct geometric realizations without worrying about whether the construction includes these edges.
- For a ramp that connects lanes x and y, optional edges to other lanes between x and y in the sequence. Edges to lanes outside that range are not allowed.

Fig. 1 depicts an example.

Definition 2. Let $\mathcal{F}$ be a collection of flips that could be made in the flipping game (a family of sets of vertices of a given graph). We define two lanes of an interchange to be *equivalent under $\mathcal{F}$* if, for every flip F in $\mathcal{F}$, either both lanes belong to F or both are omitted from F.

Lemma 3. *Let a, b, c and d, e, f be two disjoint triples of lanes such that, for a collection of flips $\mathcal{F}$, all lanes in $\{a, b, c\}$ are equivalent under $\mathcal{F}$, and all lanes in $\{d, e, f\}$ are equivalent under $\mathcal{F}$. Then, after the flips in $\mathcal{F}$ are made, the flipped interchange contains at least one two-edge lane–ramp–lane path between $\{a, b, c\}$ and $\{d, e, f\}$.*

Proof. Assume (by swapping the triples if necessary) that the lane b is before the lane e. Then these six lanes contain the four-lane subsequence a, b, e, f. If ramp be is flipped with respect to the equivalent lanes $\{a, b, c\}$, it becomes adjacent to a; otherwise it remains adjacent to b. If ramp be is flipped with respect to the equivalent lanes $\{d, e, f\}$ it becomes adjacent to f; otherwise it remains adjacent to e. In all cases this ramp connects at least one lane in $\{a, b, c\}$ to at least one lane in $\{d, e, f\}$. □

Lemma 4. *Suppose that distinct lanes a, b, and c, in an interchange of order n, are equivalent under a collection of flips $\mathcal{F}$. Then, in the flipped interchange, at least one of a, b, or c has paths of length two to at least $\frac{1}{3}\big(n - 2^{|\mathcal{F}|+1} - 3\big)$-many other lanes.*

Proof. Under the flips in $\mathcal{F}$, there are $2^{|\mathcal{F}|}$ equivalence classes of lanes. By Lemma 3, each equivalence class has at most two lanes (other than a, b, and c) that are not connected by a two-edge path to at least one of a, b, and c, because three disconnected but equivalent lanes would contradict the lemma. The total number of these disconnected vertices is at most $2^{|\mathcal{F}|+1}$; the remaining $n - 2^{|\mathcal{F}|+1} - 3$ vertices have two-edge paths to at least one of a, b, or c. Even if each were connected to exactly one of a, b, or c, and even if these connections were evenly distributed between a, b, and c, the statement of the lemma would hold. Multiple connections or uneven distribution of connections only increases the largest of the three numbers of connections among a, b, and c. □

Theorem 5. *Suppose that cops and a robber play the radius-2 flipping game with t flips per move on a graph that includes an interchange of order $n = 2^{t+3} + 3$. Then the robber can win by moving at each step (including the initial step) to a lane that maximizes the number of lanes that will be reachable after the announced flips.*

Proof. In an interchange of this size, by Lemma 4, every three equivalent lanes include one connected to $\geq 2^{t+1}$ other lanes by two-edge paths; it can reach $\geq 2^{t+1} + 1$ lanes including itself by paths of length ≤ 2. Among every $2^{t+1} + 1$ lanes, at least three are equivalent. By induction, at each step, the robber has a choice of $\geq 2^{t+1}+1$ lanes to move to, among which three are equivalent, and therefore can move to a lane that will continue to reach at least $2^{t+1} + 1$ lanes after the announced flips. □

Corollary 6. *A class of graphs that contains arbitrarily large interchanges does not have bounded flip-width.*

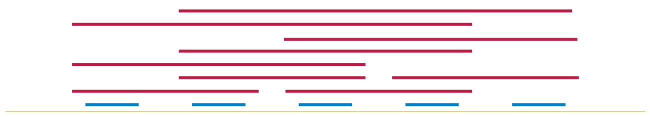


Figure 2: Representing an interchange using the intervals of an interval graph or interval containment graph.

In Appendix A we show, more strongly, that graph classes with large interchanges are not monadically dependent, a property that generalizes both classes of bounded flip-width [28] and classes that are nowhere-dense [1].

4 Geometric graphs

The geometric graphs known to have bounded flip-width include the unit interval graphs (which more strongly have bounded twin-width [4]) and the intersection graphs of disks of bounded ply in any fixed dimension (which more generally have bounded expansion [12, 24]). We prove that many other classes of geometric graphs do not have bounded flip-width, by finding large interchanges in them and applying Corollary 6.

Theorem 7. *The interval graphs, permutation graphs, circle graphs, and intersection graphs of axis-aligned line segments (no two collinear) have unbounded flip-width.*

Proof. We construct intervals representing an arbitrarily large interchange, with short disjoint intervals for each lane and long intervals spanning multiple lanes for each ramp (Fig. 2). The intersection graph of these intervals is an interval graph forming the interchange, with all optional lane–ramp edges included. The interval containment graph, having a vertex per interval and an edge whenever one interval contains another, differs only in some optional ramp–ramp edges. Interval containment graphs are the same as permutation graphs, and are a subclass of circle graphs [7]. For axis-aligned line segments, lift the ramp intervals to distinct y-coordinates, and replace the lane intervals by tall vertical segments. □

The graph classes in Theorem 7 are monadically independent [2], from which unbounded flip-width follows, but without a bound on the robber's escape radius.

Theorem 8. *The intersection graphs of axis-aligned unit squares have unbounded flip-width.*

Proof. Place the intervals of Theorem 7 on a diagonal line, scaled to have length less than $\sqrt{2}$. Represent lanes by squares below this line, intersecting the line in the given interval, and represent ramps by squares above this line (Fig. 3). The resulting unit square intersection graph may have additional lane–lane and ramp–ramp intersections, but these only create optional edges. □

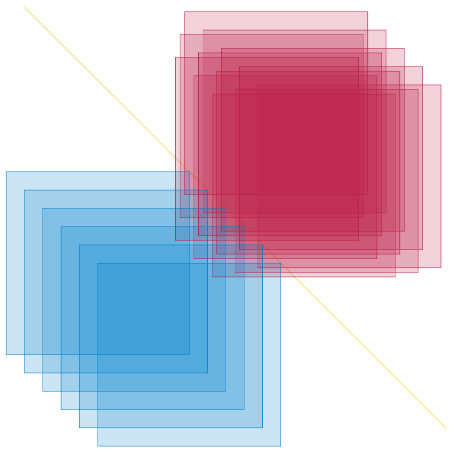


Figure 3: Representing an interchange using axis-aligned unit squares.

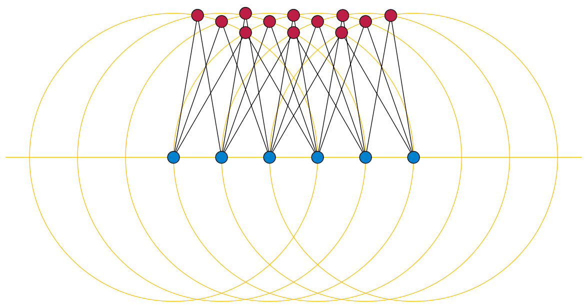


Figure 4: Representing an interchange as a unit distance graph or the center points of a unit disk graph.

Theorem 9. *The unit distance graphs and unit disk graphs have unbounded flip-width.*

Proof. For unit distance graphs, place points representing the lanes equally spaced along a line segment of length less than two in the plane, and place points representing ramps at the intersections of pairs of unit circles centered at the lane points (Fig. 4). The resulting graph may have ramp–ramp or lane–lane edges, but it will have no optional lane–ramp edges. For unit disk graphs, scale the same points by a factor of two so that unit disks centered at them will be tangent when their points are adjacent in the unit-distance graph. The resulting unit disk graph includes all possible optional lane–ramp edges, forming an interchange of the same order. □

Theorem 10. *The visibility graphs of simple polygons do not have bounded flip-width.*

Proof. Place points representing lanes on a horizontal line, and place points representing the ramps between two consecutive lanes in the same order on a parallel line above them. Place points representing the remaining ramps, between non-consecutive lanes, on a third parallel line below the lanes. Draw a triangle between each ramp

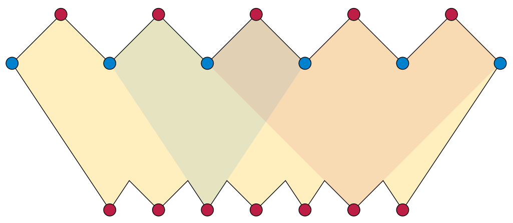


Figure 5: Representing an interchange as the visibility graph of a simple polygon.

vertex and the two lanes it should connect, and take the union of the triangles. Fill any holes formed in taking the union, keeping only the outer boundary, to form a simple polygon (Fig. 5). In the resulting polygon, each ramp still has parts of two triangle sides adjoining it, blocking its visibility from any lanes that it should not see. Within the triangle for each ramp, it can see its two lanes and any other lane between them. □

Theorem 10 also follows from the fact that simple polygon visibility graphs are monadically independent [2], which was shown using a similar construction. Furthermore, simple polygon visibility graphs are *cop-win graphs*; a single cop wins a different cop-and-robber game in which both players move along graph edges or stand still [23]. But this has no implications for flip-width; adding a universal vertex to any graph makes it cop-win but does not change the boundedness of its flip-width.

The *β-skeletons* (for $\beta \le 1$) are defined from a set of points by constructing for each pair of points a lune, the intersection of two congruent disks that cross at these points with angle $\pi - \sin^{-1}\beta$. Two points are adjacent if this lune contains no other given points [20].

Theorem 11. *For any $\beta < 1$, the β-skeletons have unbounded flip-width.*

Proof. Place vertices representing lanes and ramps on the lines $y = 0$ and $y = 1$ respectively, evenly spaced and very close to the line $x = 0$, close enough to ensure that each lane–ramp lune stays within the slab $0 \le y \le 1$. For each ramp r, place two blocking points on the line $y = 1 - \varepsilon$ for suitably small ε, close enough to r to avoid all lunes from other ramps. These blocking points should be just outside the two lunes connecting r to its two lanes, one to the left and one to the right, so that any lune connecting r to a lane outside of its range of lanes contains one of the blocking points and is non-empty. The resulting β-skeleton forms an interchange with all optional lane–ramp edges. □

Observation 12. *The graph of a d-dimensional hypercube contains an interchange of order d.*

Proof. This is the graph of subsets of a d-element set, adjacent when they differ by one element. Let lanes be singletons and ramps be two-element sets. □

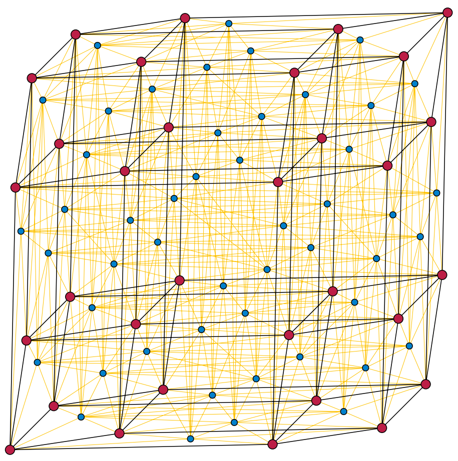


Figure 6: A 5-dimensional hypercube graph as an induced subgraph of a rectangle of influence graph.

Rectangle of influence graphs connect pairs of points in the plane when their bounding box contains no other points [18, 22]. Sources vary on how to treat points on the boundary of this bounding box, but that can be avoided using point sets with no equal coordinates.

Theorem 13. *Rectangle of influence graphs induce high-dimension hypercubes and have unbounded flip-width.*

Proof. We recursively construct integer points with distinct coordinates whose rectangle of influence graphs contain arbitrarily large induced hypercubes. As a base case, the two points $(0, 0)$ and $(1, 1)$ give a one-dimensional hypercube graph. If X_{d-1} is defined in this way, with subset Y_{d-1} inducing a $(d-1)$-dimensional hypercube, construct X_d by placing side to side the following three sets: (1) a copy of X_{d-1} scaled vertically by a factor of three; (2) a copy of Y_{d-1} scaled by the same factor, offset vertically by two units (and missing its topmost point), and (3) another scaled copy of X_{d-1}, offset vertically by one unit. Choose Y_d to be the copies of Y_{d-1} within the first and third of these three sets. Scaling does not affect the hypercube graphs within these copies. The middle copy of Y_{d-1} blocks all empty rectangles stretching from the first copy to the third copy, except those between corresponding pairs of points, so Y_d induces a d-dimensional hypercube graph. □

Fig. 6 illustrates five levels of the recursive construction of Theorem 13, with another (cosmetic) step that compacts the coordinates to use consecutive integers.

Theorem 14. *The graphs of four-dimensional convex polytopes, and of three-dimensional Euclidean Delaunay triangulations, have unbounded flip-width.*

Proof. For 4-polytopes, consider the barycentric subdivisions of neighborly polytopes. The graph of a neighborly polytope is complete, and barycentric subdivision preserves realizability as a convex polytope [13]. The barycentric subdivision replaces the edges of the complete graph by disjoint two-edge paths. The original vertices of the complete graph form the lanes, and the subdivision points of these paths form the ramps, of an interchange, whose order equals the number of vertices in the neighborly polytope.

For Delaunay triangulations, we do not use interchanges; instead we rely on a result of Toruńczyk that weakly sparse graphs (that is, graphs with no $K_{t,t}$ subgraph for some t) have bounded flip-width if and only if they have bounded expansion [28]. To construct a Delaunay triangulation that does not have bounded flip-width, we begin with the convex hull of certain points in $\mathbb{R}^4$ (coordinatized by pairs of complex numbers), the union of the following three sets of points on a unit sphere, for a given even integer parameter n:

- The n points $(e^{2\pi i/n}, 0)$ for integer i, $0 \le i < n$.
- The n points $(0, e^{2\pi j/n})$ for integer j, $0 \le j < n$.
- The n^2 points $(e^{2\pi i/n}/\sqrt{2}, e^{2\pi j/n}/\sqrt{2})$ for i and j in the same range.

The points with both coordinates nonzero form a square grid on the flat torus $\{(x, y) \mid |x| = |y| = 1/\sqrt{2}\}$, and the other two subsets of points form n-gons (not faces of the convex hull) in the planes $x = 0$ and $y = 0$. Each edge of an n-gon is parallel to the family of edges connecting a ring of squares on the torus, and the facets of the convex hull are warped triangular pyramids connecting an n-gon edge to one of these parallel squares. The subgraph induced in the graph of this polytope by the vertices for which i and j are both even is the subdivision of a complete bipartite graph $K_{n/2,n/2}$. This subdivision has no $K_{2,2}$ subgraph and does not have bounded expansion, so it does not have bounded flip-width.

To transform this inscribed 4-polytope into a Delaunay triangulation, we apply a stereographic projection whose pole is the center point of one of the squares on the torus. On the unit sphere in $\mathbb{R}^4$, this pole belongs to only two of the circumspheres of the facets, for the two facets meeting at this square. Stereographic projection preserves spheres on the unit sphere, so the empty spheres of all of the other facets project to empty spheres for the corresponding set of six points in $\mathbb{R}^3$; that is, these six points form a prism-shaped cell in the Delaunay complex of the projected points. Each edge of the 4-polytope is part of one of these Delaunay cells, so the Delaunay graph of the projected points is the same as that of the complex. Perturbing the points to form a Delaunay triangulation, and keeping only the points for which i and j are both even, again produces the subdivision of a complete bipartite graph, as the subgraph of a three-dimensional Delaunay triangulation. □

5 Radius-1 flip-width

Our escape strategy for the robber through interchanges involves the robber taking two steps per move. It is natural to ask whether this can be strengthened to allow escapes of only one edge per move for the same classes of geometric graphs. That is, do these geometric graphs have bounded or unbounded radius-1 flip-width?

In the treewidth game, radius-1 corresponds to *degeneracy*, where a graph has degeneracy $\leq d$ if and only if its vertices can be ordered so that each vertex has $\leq d$ earlier neighbors. The radius-1 treewidth game can be won by $d+1$ cops, who always play on the current vertex of the robber and its earlier neighbors, forcing the robber to move later in the ordering. On the other hand, if the degeneracy is greater than d then the graph has a $(d+1)$-core, an induced subgraph with minimum degree $d+1$, within which the robber is safe: no matter where the cops move next, there will be an unoccupied vertex for the robber within one step [28].

For the radius-1 flip-width game, Toruńczyk identifies a corresponding concept to a core, which we call a Δ-*diverse subgraph*. This is an induced subgraph in which the open neighborhoods of each two vertices differ by at least Δ vertices. As Toruńczyk proves, a family of graphs whose graphs contain Δ-diverse subgraphs, for arbitrarily large Δ, has unbounded radius-1 flip-width [28]. For cops that make t flips per move, the robber can escape by staying within a 2^{t+1}-diverse subgraph and moving to a vertex v in this subgraph such that, after the announced flips, v will have at least 2^t neighbors. These neighbors, and v itself, form a set of 2^t+1 vertices within which the robber can move, some two of which (say x and y) will be equivalent after the cops' flips. Each vertex adjacent to exactly one of x and y will remain adjacent to exactly one after the flips, so one of x and y will have $\geq 2^t$ neighbors, enough to continue the same strategy.

Slightly more generally, define a (Δ,χ)-diverse subgraph for a subset of vertices in a given graph and an (improper) χ-coloring of those vertices, keeping all properly colored edges and removing all improperly colored ones, with diversity measured between vertices of the same color. The robber escapes by staying in a $(2^{t+1}(\chi-1)+2,\chi)$-diverse subgraph and moving to a vertex with at least $2^t(\chi-1)+1$ differently-colored neighbors. Some two neighbors will have the same color and be treated equivalently by all flips, and one of these two will have enough neighbors in the next move.

Lemma 15. *An order-n interchange with all optional lane–ramp edges has an $(\Omega(n^{1/3}),2)$-diverse subgraph.*

Proof. Number the lanes of the interchange from 0 to $n-1$, and choose a parameter $k\approx n^{1/3}$. Form a two-colored subgraph (colored by lanes and ramps) with all lanes and a subset of ramps, the ramps for lanes x and y with $x<y$ that meet the following two conditions: (1) $|y-x|\geq k$, and (2) $x\equiv ky \bmod k^2+1$. The second condition leaves $\Theta(n^{1/3})$ ramps starting or ending at each lane, enough so every two lanes have diverse neighborhoods.

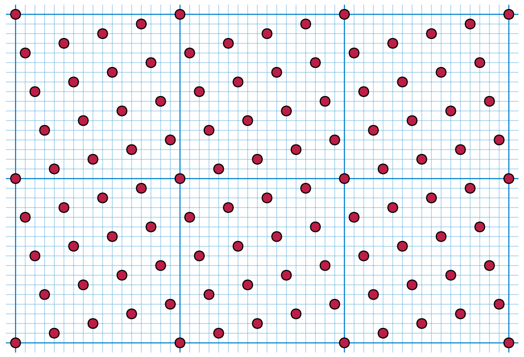


Figure 7: The integer points (x,y) with $x\equiv ky$ mod k^2+1 (shown for $k=4$) form a tilted grid in which all pairs of points have L_1 distance at least $k+1$.

Two ramps with disjoint ranges of lanes are distinguished by all their neighbors, of which there are $\Omega(n^{1/3})$ by the first condition. Two ramps with overlapping ranges of lanes $[x,y]$ and $[x',y']$ are distinguished by the lanes between x and x', and the lanes between y and y'; there are $|x-x'|+|y-y'|$ such lanes. The condition that $x\equiv ky \bmod k^2+1$ defines a subset of the integer grid in the form of a tilted square grid in which all pairs of points are at L_1 distance at least $k+1$ (Fig. 7), so the number of lanes that are neighbors of only one of the two ramps is also at least $k+1=\Omega(n^{1/3})$. □

Corollary 16. *Interval graphs, permutation graphs, circle graphs, intersection graphs of axis-aligned unit squares, unit disk graphs, visibility graphs of simple polygons, and β-skeletons with $\beta<1$ have unbounded radius-1 flip-width.*

Observation 17. *The graph of a d-dimensional hypercube is $(2d-2)$-diverse.*

Proof. It is d-regular, and the neighborhoods of any two vertices can share at most two neighbors. □

Corollary 18. *Unit distance graphs, graphs of 4-polytopes, and rectangle of influence graphs have unbounded radius-1 flip-width.*

Proof. The d-dimensional hypercube graphs can be drawn as unit distance graphs by linear projections that map the basis vectors of $\mathbb{R}^d$ to generic unit vectors in the plane [8]. They are the graphs of the 4-dimensional *neighborly cubical polytopes* [19]. For rectangle of influence graphs, the result follows from Theorem 13 □

We do not know whether three-dimensional Delaunay triangulations have bounded radius-1 flip-width; we leave this as open for future research.

References

[1] Hans Adler and Isolde Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *Eur. J. Comb.*, 36:322–330, 2014. doi:10.1016/j.ejc.2013.06.048.

[2] Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-Width VIII: Delineation and win-wins. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7–9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.9.

[3] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In Dániel Marx, editor, *Proceedings of the 2021 ACM–SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 1977–1996. SIAM, 2021. doi:10.1137/1.9781611976465.118.

[4] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12–16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.35.

[5] Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20–24, 2022*, pages 924–937, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520037.

[6] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):A3:1–A3:46, 2022. doi:10.1145/3486655.

[7] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey.* SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999. doi:10.1137/1.9780898719796.

[8] Peter Brass. Erdős distance problems in normed spaces. *Comput. Geom. Theory & Appl.*, 6(4):195–214, 1996. doi:10.1016/0925-7721(95)00019-4.

[9] Samuel Braunfeld and Michael C. Laskowski. Characterizations of monadic NIP. *Trans. Amer. Math. Soc. Ser. B*, 8:948–970, 2021. doi:10.1090/btran/94.

[10] Jan Dreier, Nikolas Mählmann, and Sebastian Siebertz. First-order model checking on structurally sparse graph classes. Electronic preprint arxiv:2302.03527, 2023.

[11] Zdeněk Dvořák, Mirna Džamonja, Agelos Georgakopoulos, Jan Obdržálek, Patrice Ossona de Mendez, Sylvain Schmitz, Szymon Toruńczyk, and Jan Volec. Open problems. In *Workshop on Algorithms, Logic and Structure*. University of Warwick, December 12–14 2016. Online web site, accessed March 19, 2023. URL: https://warwick.ac.uk/fac/sci/maths/people/staff/daniel_kral/alglogstr/openproblems.pdf.

[12] Zdeněk Dvořák and Sergey Norin. Strongly sublinear separators and polynomial expansion. *SIAM J. Discrete Math.*, 30(2):1095–1101, 2016. doi:10.1137/15M1017569.

[13] G. Ewald and G. C. Shephard. Stellar subdivisions of boundary complexes of convex polytopes. *Math. Ann.*, 210:7–16, 1974. doi:10.1007/BF01344542.

[14] Jakub Gajarský, Petr Hliněný, Daniel Lokshtanov, Jan Obdrzálek, Sebastian Ordyniak, M. S. Ramanujan, and Saket Saurabh. FO model checking on posets of bounded width. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17–20 October, 2015*, pages 963–974. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.63.

[15] Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Daniel Lokshtanov, and M. S. Ramanujan. A new perspective on FO model checking of dense graph classes. *ACM Trans. Comput. Log.*, 21(4):A28:1–A28:23, 2020. doi:10.1145/3383206.

[16] Robert Ganian, Petr Hliněný, Daniel Král, Jan Obdržálek, Jarett Schwartz, and Jakub Teska. FO model checking of interval graphs. *Log. Methods Comput. Sci.*, 11(4):1–20, 2015. doi:10.2168/LMCS-11(4:11)2015.

[17] Petr Hliněný, Filip Pokrývka, and Bodhayan Roy. FO model checking on geometric graphs. *Comput. Geom. Theory & Appl.*, 78:1–19, 2019. doi:10.1016/j.comgeo.2018.10.001.

[18] Manabu Ichino and Jack Sklansky. The relative neighborhood graph for mixed feature variables. *Pattern Recognition*, 18(2):161–167, 1985. `doi:10.1016/0031-3203(85)90040-8`.

[19] M. Joswig and Günter M. Ziegler. Neighborly cubical polytopes. *Discrete Comput. Geom.*, 24(2-3):325–344, 2000. `doi:10.1007/s004540010039`.

[20] David G. Kirkpatrick and John D. Radke. A framework for computational morphology. In *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 217–248. North-Holland, Amsterdam, 1985.

[21] James R. Lee. Separators in region intersection graphs. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9–11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 1:1–1:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.ITCS.2017.1`.

[22] Giuseppe Liotta, Anna Lubiw, Henk Meijer, and Sue H. Whitesides. The rectangle of influence drawability problem. *Comput. Geom. Theory & Appl.*, 10(1):1–22, 1998. `doi:10.1016/S0925-7721(97)00018-7`.

[23] Anna Lubiw, Jack Snoeyink, and Hamideh Vosoughpour. Visibility graphs, dismantlability, and the cops and robbers game. *Comput. Geom. Theory & Appl.*, 66:14–27, 2017. `doi:10.1016/j.comgeo.2017.07.001`.

[24] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, January 1997. `doi:10.1145/256292.256294`.

[25] Jaroslav Nešetřil and Patrice Ossona de Mendez. 5.5 Classes with bounded expansion. In *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*, pages 104–107. Springer, 2012. `doi:10.1007/978-3-642-27875-4`.

[26] Jaroslav Nesetril, Patrice Ossona de Mendez, and Sebastian Siebertz. Structural properties of the first-order transduction quasiorder. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14–19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.CSL.2022.31`.

[27] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Combinatorial Theory, Ser. B*, 58(1):22–33, 1993. `doi:10.1006/jctb.1993.1027`.

[28] Szymon Toruńczyk. Flip-width: cops and robber on dense graphs. Electronic preprint arxiv:2302.00352, 2023.

A Monadic dependence

In this appendix we show that graph classes that include arbitrarily large interchanges are monadically independent. This in particular implies that they do not have bounded flip-width, but unlike Theorem 5 it does not directly bound the radius r at which the robber can win.

Here, *monadic dependence* is a general notion for logical relations of arbitrary arity based on the concept of a *transduction*, a method of representing one logical structure by a system of first-order formulas over a finite system of monadic predicates, or equivalently over a finite coloring, of a second structure [9]. When applied to the first-order theory of graphs, a transduction is a mapping from vertex-colored graphs to graphs, defined by logical predicates that describe which pairs of vertices are adjacent in the image graph, and which vertices of the starting graph correspond to vertices in the image graph [26]. A family of graphs $\mathcal{F}$ is monadically dependent (also written as "monadically NIP") if every transduction on the graphs in $\mathcal{F}$ is incomplete: there is some target graph that it does not produce, regardless of which graph in $\mathcal{F}$ and which coloring of that graph it is applied to. On the other hand, $\mathcal{F}$ is monadically independent ("not monadically NIP") if it is possible to find a "universal" transduction, one that maps colorings of graphs in $\mathcal{F}$ to all possible graphs.

It is conjectured that, among hereditary graph families, the monadically dependent families are exactly the ones for which first-order model checking is fixed-parameter tractable [11, 15]. For hereditary classes of ordered graphs, and logical properties that can make use of this ordering, being monadically dependent is equivalent to having bounded twin-width [5]. Without the assumption of an ordering, bounded flip-width implies monadically dependence, and a weakening of bounded flip-width, *almost bounded flip-width*, is conjectured to be equivalent to monadic dependence [28].

In this section we use a slightly stronger definition of interchanges that includes ramps between all pairs of lanes, instead of omitting the outermost lanes. An interchange under this stronger definition satisfies the weaker definition (where fewer ramps are required). In the other direction, an interchange for the weaker definition can be converted to one for the stronger definition, with two fewer lanes, by omitting the two outermost lanes.

We first observe that every huge interchange contains, as an induced subgraph, a large interchange in which either all optional ramp-lane edges are present, or a large interchange in which no optional ramp-lane edges are present. Call those two types of interchanges *dense interchanges* and *sparse interchanges*, respectively.

Lemma 19. *If a class of graphs contains arbitrarily large interchanges, it contains either arbitrarily large dense interchanges or arbitrarily large sparse interchanges.*

Proof. This follows from applying standard methods of Ramsey theory to a 2-colored 3-uniform hypergraph on triples of lanes, with each triple given color 1 if an optional edge connects the middle of the three lanes to the ramp between the two outer lanes, and color 0 if the optional edge is not present. By Ramsey's theorem, we can extract a large subset of the lanes that induces a monochromatic sub-hypergraph. The selected lanes, together with the ramps associated with pairs of selected lanes, then form a large regular interchange: if they all have color 1, the result is a dense interchange, and if they all have color 0, the result is a sparse interchange. □

The following result is from Szymon Toruńczyk (personal communication):

Theorem 20. *A graph class that contains arbitrarily large interchanges as induced subgraphs is monadically independent.*

Proof. By Lemma 19, we can assume either that there are arbitrarily large sparse interchanges, or that there are arbitrarily large dense interchanges. We show that in each of these cases, the class is monadically independent.

We first argue that any class containing arbitrarily large sparse interchanges is monadically independent. We can represent an arbitrary n-vertex graph G in a colored sparse interchange with at least n lanes, by identifying the vertices of G with a subset of the lanes, and for each ramp associated with a pair uv of vertices of G, marking this ramp with a special color if and only if u and v are adjacent in G. Formally, describe this marking by a monadic predicate $A(z)$ that is true of these marked ramps and false for all other vertices. Then adjacency in the original graph G can be recovered from the obtained colored interchange, by a first-order formula

$$\phi(x, y) \equiv \exists z\big(x \sim z \wedge y \sim z \wedge A(z)\big)$$

(where adjacency in G is represented by the binary predicate $\sim$), expressing that there is a vertex z that is adjacent to x and y and has the special marking. The formula ϕ does not depend on the selected graph G. It follows that the formula ϕ can be used to define a transduction that may produce an arbitrary graph G from any sufficiently large sparse interchange, by first coloring its vertices using a single special color, then applying the formula $\phi(x, y)$ to define a new edge relation, and finally, taking an induced subgraph to restrict only to the lanes of the interchange that correspond to the vertices of G. Thus, every class transduces the class of all graphs, and is therefore monadically independent.

We now argue that every class containing arbitrarily large dense interchanges is monadically independent, by showing that such a class transduces a class of arbitrarily large sparse interchanges. Since transductions can be composed, this implies that the former class is monadically independent.

Given a dense interchange, color all the ramps using a special color, denoted by the monadic predicate $R(v)$, and color the first lane (in the order on lanes) with another color, denoted by the monadic predicate $F(v)$. We can recover the order on two lanes x and y using a first-order formula

$$\begin{aligned}\psi(x,y) \equiv& \\ &\big(F(x) \wedge x \neq y\big) \vee \\ &\exists w \exists z \big(F(w) \wedge R(z) \wedge w \sim z \wedge x \sim z \wedge \neg(y \sim z)\big).\end{aligned}$$

Namely, a lane x is smaller than a lane y if and only if either x is the first lane, or there a ramp z which is adjacent to the first lane and to x but not to y. Since ramps and the first lane are marked with a special color, this can be expressed using a fixed first-order formula (not depending on the interchange), that can use the colors.

Now, we can identify the two lanes x, y to which a ramp z is associated with, using a first-order formula. Namely, in a sparse interchange, a ramp z should be associated with exactly the smallest lane, and the largest lane among its neighboring lanes. Therefore, z is associated with a lane x if and only if z is adjacent to x and there do not exist lanes u and v, smaller and larger than x, respectively, that are also adjacent to z. This can be expressed by a first-order formula

$$\begin{aligned}\gamma(x,z) \equiv \neg R(x) \wedge R(z) \wedge x \sim z \wedge& \\ \neg\exists u \exists v &\begin{pmatrix}\neg R(u) \wedge \neg R(v) \wedge \\ u \sim z \wedge v \sim z \wedge \\ \psi(u,x) \wedge \psi(x,v)\end{pmatrix}.\end{aligned}$$

that uses the colors, and does not depend on the considered interchange. Thus, the formula $\gamma(x,z)$ defines the edges of a sparse interchange, of the same size as the original dense interchange.

Hence, every class that contains arbitrarily large dense interchanges as induced subgraphs transduces a class that contains arbitrarily large sparse interchanges, and is therefore monadically independent. □

This applies, in particular, to all the classes of geometric graphs for which we have constructed large interchanges. Our proof that three-dimensional Delaunay triangulations have unbounded flip-width is an exception, as it does not use interchanges. However, in this case we have constructed weakly sparse Delaunay triangulations that are not nowhere-dense, so it follows from known results that they are monadically independent.

Graph Mover's Distance: An Efficiently Computable Distance Measure for Geometric Graphs

Sushovan Majhi*

Abstract

Many applications in pattern recognition represent patterns as a geometric graph. The geometric graph distance (GGD) has recently been studied in [13] as a meaningful measure of similarity between two geometric graphs. Since computing the GGD is known to be $\mathcal{NP}$–hard, the distance measure proves an impractical choice for applications. As a computationally tractable alternative, we propose in this paper the Graph Mover's Distance (GMD), which has been formulated as an instance of the earth mover's distance. The computation of the GMD between two geometric graphs with at most n vertices takes only $O(n^3)$-time. Alongside studying the metric properties of the GMD, we investigate the stability of the GGD and GMD. The GMD also demonstrates extremely promising empirical evidence at recognizing letter drawings from the `LETTER` dataset [18].

1 Introduction

Graphs have been a widely accepted object for providing structural representation of patterns involving relational properties. While hierarchical patterns are commonly reduced to a string [7] or a tree representation [6], non-hierarchical patterns generally require a graph representation. The problem of pattern recognition in such a representation then requires quantifying (dis-)similarity between a query graph and a model or prototype graph. Defining a relevant distance measure for a class of graphs has been studied for almost five decades now and has a myriad of applications including chemical structure matching [21], fingerprint matching [16], face identification [11], and symbol recognition [12].

Depending on the class of graphs of interest and the area of application, several methods have been proposed. Graph isomorphisms [5] or subgraph isomorphisms can be considered.

These, however, cannot cope with (sometimes minor) local and structural deformations of the two graphs. To address this issue, several alternative distance measures have been studied. We particularly mention *edit distance* [20, 9] and *inexact matching distance* [3].

*School of Information, University of California, Berkeley, USA, `smajhi@berkeley.edu`

Although these distance measures have been battle-proven for attributed graphs (i.e., combinatorial graphs with finite label sets), the formulations seem inadequate in providing meaningful similarity measures for geometric graphs.

A geometric graph belongs to a special class of attributed graphs having an embedding into a Euclidean space $\mathbb{R}^d$, where the vertex labels are inferred from the Euclidean locations of the vertices and the edge labels are the Euclidean lengths of the edges.

In the last decade, there has been a gain in practical applications involving comparison of geometric graphs, such as road-network or map comparison [1], detection of chemical structures using their spatial bonding geometry, etc. In addition, large datasets like [18] are being curated by pattern recognition and machine learning communities.

1.1 Related Work and Our Contribution

We are inspired by the recently developed geometric graph distance (GGD) in [4, 13]. Although the GGD succeeds to be a relevant distance measure for geometric graphs, its computation, unfortunately, is known to be $\mathcal{NP}$-hard. Our motivation stems from applications that demand an efficiently computable measure of similarity for geometric graphs. The formulation of our graph mover's distance is based on the theoretical underpinning of the GGD. The GMD provides a meaningful yet computationally efficient similarity measure between two geometric graphs.

In Section 2, we revisit the definition of the (GGD) to investigate its stability under Hausdorff perturbation. Section 3 is devoted to the study of the GMD. The GMD has been shown to render a *pseudo*-metric on the class of (ordered) geometric graphs. Finally, we apply the GMD to classify letter drawings in Section 4. Our experiment involves matching each of 2250 test drawings, modeled as geometric graphs, to 15 prototype letters from the English alphabet. For the drawings with `LOW` distortion, the correct letter has been found among the top 3 matches at a rate of 98.93%, where the benchmark accuracy is 99.6% obtained using a k-nearest neighbor classifier (k-NN) with the graph edit distance [3].

2 Geometric Graph Distance (GGD)

We first formally define a geometric graph. Throughout the paper, the dimension of the ambient Euclidean space is denoted by $d \geq 1$. We also assume that the cost coefficients C_V and C_E are positive constants.

Definition 2.1 (Geometric Graph) *A geometric graph of $\mathbb{R}^d$ is a (finite) combinatorial graph $G = (V^G, E^G)$ with vertex set $V^G \subset \mathbb{R}^d$, and the Euclidean straight-line segments $\{\overline{ab} \mid (a,b) \in E^G\}$ intersect (possibly) at their endpoints.*

We denote the set of all geometric graphs of $\mathbb{R}^d$ by $\mathcal{G}(\mathbb{R}^d)$. Two geometric graphs $G = (V^G, E^G)$ and $H = (V^H, E^H)$ are said to be *equal*, written $G = H$, if and only if $V^G = V^H$ and $E^G = E^H$. We make no distinction between a geometric graph $G = (V^G, E^G)$ and its *geometric realization* as a subset of $\mathbb{R}^d$; an edge $(u,v) \in E^G$ can be identified as the line-segment $\overline{uv}$ in $\mathbb{R}^d$, and its length by the Euclidean length $|\overline{uv}|$.

Following the style of [13], we first revisit the definition of GGD. The definition uses the notion of an inexact matching. In order to denote a deleted vertex and a deleted edge, we introduce the *dummy vertex* ϵ_V and the *dummy edge* ϵ_E, respectively.

Definition 2.2 (Inexact Matching) *Let $G, H \in \mathcal{G}(\mathbb{R}^d)$ be two geometric graphs. A relation $\pi \subseteq (V^G \cup \{\epsilon_V\}) \times (V^H \cup \{\epsilon_V\})$ is called an (inexact) matching if for any $u \in V^G$ (resp. $v \in V^H$) there is exactly one $v \in V^H \cup \{\epsilon_V\}$ (resp. $u \in V^G \cup \{\epsilon_V\}$) such that $(u,v) \in \pi$.*

The set of all matchings between graphs G, H is denoted by $\Pi(G,H)$. Intuitively, a matching π is a relation that covers the vertex sets V^G, V^H exactly once. As a result, when restricted to V^G (resp. V^H), a matching π can be expressed as a map $\pi : V^G \to V^H \cup \{\epsilon_V\}$ (resp. $\pi^{-1} : V^H \to V^G \cup \{\epsilon_V\}$). In other words, when $(u,v) \in \pi$ and $u \neq \epsilon_V$ (resp. $v \neq \epsilon_V$), it is justified to write $\pi(u) = v$ (resp. $\pi^{-1}(v) = u$). It is evident from the definition that the induced map

$$\pi : \{u \in V^G \mid \pi(u) \neq \epsilon_V\} \to \{v \in V^H \mid \pi^{-1}(v) \neq \epsilon_V\}$$

is a bijection. For edges $e = (u_1, u_2) \in E^G$ and $f = (v_1, v_2) \in E^H$, we introduce the short-hand $\pi(e) := (\pi(u_1), \pi(u_2))$ and $\pi^{-1}(f) := (\pi^{-1}(v_1), \pi^{-1}(v_2))$.

Another perspective of π is to view it as a matching between portions of G and H, (possibly) after applying some edits on the two graphs. For example, $\pi(u) = \epsilon_V$ (resp. $\pi^{-1}(v) = \epsilon_V$) encodes deletion of the vertex u from G (resp. v from H), whereas $\pi(e) = \epsilon_E$ (resp. $\pi^{-1}(f) = \epsilon_E$) encodes deletion of the edge e from G (resp. f from H). Once the above deletion operations have been performed on the graphs, the resulting subgraphs of G and H become isomorphic, which are finally matched by translating the remaining vertices u to $\pi(u)$. Now, the cost of the matching π is defined as the total cost for all of these operations:

Definition 2.3 (Cost of a Matching) *Let $G, H \in \mathcal{G}(\mathbb{R}^d)$ be geometric graphs and $\pi \in \Pi(G,H)$ an inexact matching. The cost of π, is* $\text{Cost}(\pi) =$

$$\underbrace{\sum_{\substack{u \in V^G \\ \pi(u) \neq \epsilon_V}} C_V |u - \pi(u)|}_{\textit{vertex translations}} + \underbrace{\sum_{\substack{e \in E^G \\ \pi(e) \neq \epsilon_E}} C_E \big| |e| - |\pi(e)| \big|}_{\textit{edge translations}} + \underbrace{\sum_{\substack{e \in E^G \\ \pi(e) = \epsilon_E}} C_E |e|}_{\textit{edge deletions}} + \underbrace{\sum_{\substack{f \in E^H \\ \pi^{-1}(f) = \epsilon_E}} C_E |f|}_{\textit{edge deletions}}. \tag{1}$$

Definition 2.4 (GGD) *For geometric graphs $G, H \in \mathcal{G}(\mathbb{R}^d)$, their geometric graph distance,* $\text{GGD}(G,H)$, *is*

$$\text{GGD}(G,H) \overset{def}{=} \min_{\pi \in \Pi(G,H)} \text{Cost}(\pi).$$

2.1 Stability of GGD

A distance measure is said to be *stable* if it does not change much if the inputs are *perturbed* only slightly. Usually, the change is expected to be bounded above by the amount of perturbation inflicted on the inputs. The perturbation is measured under a suitable choice of metric. In the context of geometric graphs, it is natural to wonder if the GGD is stable under the Hausdorff distance between two graphs. To our disappointment, we can easily see for the graphs shown in Fig. 1 that the GGD is positive, whereas the Hausdorff distance between their realizations is zero. So, the Hausdorff distance between the graphs can not bound their GGD from above.

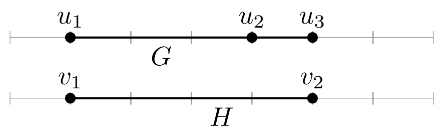


Figure 1: The graphs G (top) and H (bottom) are embedded in the real line; the distance between consecutive ticks is 1 unit. The Hausdorff distance between G and H is zero, however $\text{GGD}(G,H) = C_V + C_E$ is non-zero. The optimal matching is given by $\pi(u_1) = v_1$, $\pi(u_2) = v_2$, and $\pi(u_3) = \epsilon_V$.

One might think that the GGD is stable when the Hausdorff distance only between the vertices is considered. However, the graphs in Fig. 2 indicate otherwise.

Under strong requirements, however, it is not difficult to prove the following result on the stability of GGD under the Hausdorff distance.

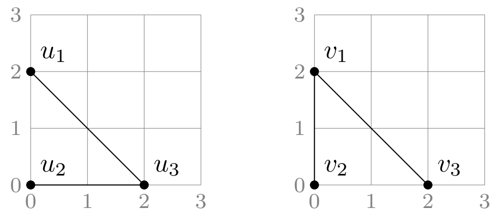


Figure 2: For the graphs $G, H \in \mathcal{G}(\mathbb{R}^2)$, the Hausdorff distance between the vertex sets is zero, however $\text{GGD}(G, H) = 4C_E$ is non-zero. The optimal matching is given by $\pi(u_1) = v_1$, $\pi(u_3) = v_3$, $\pi(u_2) = \epsilon_V$, and $\pi^{-1}(v_2) = \epsilon_V$.

Theorem 1 (Hausdorff Stability of GGD) *Let $G, H \in \mathcal{G}(\mathbb{R}^d)$ be geometric graphs with a graph isomorphism $\pi : V^G \to V^H$. If $\delta > 0$ is such that $|u - \pi(u)| \le \delta$ for all $u \in V^G$, then*

$$\text{GGD}(G, H) \le C_V |V^G| \delta.$$

Proof. The given graph isomorphism π is a bijective mapping between the vertices of G and H. So, $\pi \in \Pi(G, H)$, i.e., it defines an inexact matching. Since π is a graph isomorphism, it does not delete any vertex or edge. More formally, for all $u \in V^G$ and $v \in V^H$, we have $\pi(u) \ne \epsilon_V$ and $\pi^{-1}(v) \ne \epsilon_V$, respectively. Also, for all $e \in E^G$ and $f \in E^H$, we have $\pi(e) \ne \epsilon_E$ and $\pi^{-1}(f) \ne \epsilon_E$, respectively. From (1), the cost

$$\text{Cost}(\pi) = \sum_{u \in V^G} C_V |u - \pi(u)| \le C_V |V^G| \delta.$$

So, $\text{GGD}(G, H) \le \text{Cost}(\pi) \le C_V |V^G| \delta$. □

3 Graph Mover's Distance (GMD)

We define the *Graph Mover's Distance* for two ordered geometric graphs. A geometric graph is called *ordered* if its vertices are ordered or indexed. In that case, we denote the vertex set as a (finite) sequence $V^G = \{u_i\}_{i=1}^m$. Let us denote by $\mathcal{G}^O(\mathbb{R}^d)$ the set of all ordered geometric graphs of $\mathbb{R}^d$. The formulation of the GMD uses the framework known as the earth mover's distance (EMD).

3.1 Earth Mover's Distance (EMD)

The EMD is a well-studied distance measure between weighted point sets, with many successful applications in a variety of domains; for example, see [8, 10, 17, 19]. The idea of the EMD was first conceived by Monge [14] in 1781, in the context of transportation theory. The name "earth mover's distance" was coined only recently, and is well-justified due to the following analogy. The first weighted point set can be thought of as piles of earth (dirt) lying on the point sites, with the weight of a site indicating the amount of earth; whereas, the other point set as pits of volumes given by the corresponding weights. Given that the total amount of earth in the piles equals the total volume of the pits, the EMD computes the least (cumulative) cost needed to fill all the pits with earth. Here, a unit of cost corresponds to moving a unit of earth by a unit of "ground distance" between the pile and the pit.

The EMD can be cast as a transportation problem on a bipartite graph, which has several efficient implementations, e.g., the network simplex algorithm [2, 15]. Let the weighted point sets $P = \{(p_i, w_{p_i})\}_{i=1}^m$ and $Q = \{(q_j, w_{q_j})\}_{j=1}^n$ be a set of suppliers and a set of consumers, respectively. The weight w_{p_i} denotes the total supply of the supplier p_i, and w_{q_j} the total demand of the consumer q_j. The matrix $[d_{i,j}]$ is the matrix of ground distances, where $d_{i,j}$ denotes the cost of transporting a unit of supply from p_i to q_j. We also assume the *feasibility condition* that the total supply equals the total demand:

$$\sum_{i=1}^m w_{p_i} = \sum_{j=1}^n w_{q_j} \ . \tag{2}$$

A *flow* of supply is given by a matrix $[f_{i,j}]$ with $f_{i,j}$ denoting the units of supply transported from p_i to q_j. We want to find a flow that minimizes the overall cost

$$\sum_{i=1}^m \sum_{j=1}^n f_{i,j} d_{i,j}$$

subject to:

$$f_{i,j} \ge 0 \text{ for any } i = 1, \dots, m \text{ and } j = 1, \dots, n \tag{3}$$

$$\sum_{j=1}^n f_{i,j} = w_i \text{ for any } i = 1, \dots, m \tag{4}$$

$$\sum_{i=1}^m f_{i,j} = w_j \text{ for any } j = 1, \dots, n, \tag{5}$$

Constraint (3) ensures a flow of units from P to Q, and not vice versa; constraint (4) dictates that a supplier must send all its supply—not more or less; constraint (5) guarantees that the demand of every consumer is exactly fulfilled.

The *earth mover's distance* (EMD) is then defined by the cost of the optimal flow. A solution always exists, provided condition (2) is satisfied. The weights and the ground distances can be chosen to be any non-negative numbers. However, we choose them appropriately in order to solve our graph matching problem.

3.2 Defining the GMD

Let $G, H \in \mathcal{G}^O(\mathbb{R}^d)$ be two ordered geometric graphs of $\mathbb{R}^d$ with $V^G = \{u_i\}_{i=1}^m$ and $V^H = \{v_j\}_{j=1}^n$. For

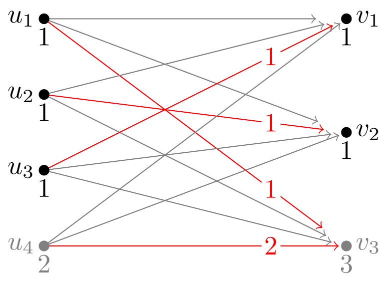


Figure 3: The bipartite network used by the GMD is shown for two ordered graphs G, H with vertex sets $V^G = \{u_1, u_2, u_3\}$ and $V^H = \{v_1, v_2\}$, respectively. The dummy nodes u_4 for G and v_3 for H, respectively, have been shown in gray. Below each node, the corresponding weights are shown. A particular flow has been depicted here. The gray edges do not transport anything. A red edge has a non-zero flow with the transported units shown on them.

each $i = 1, \ldots, m$, let E_i^G denote the (row) m–vector containing the lengths of (ordered) edges incident to the vertex u_i of G. More precisely, the

$$k\text{th element of } E_i^G = \begin{cases} |e_{i,k}^G|, \text{ if } e_{i,k}^G := (u_i, u_k) \in E^G \\ 0, \text{ otherwise.} \end{cases}$$

Similarly, for each $j = 1, \ldots, n$, we define E_j^H to be the (row) n–vector with the

$$k\text{th element of } E_j^H = \begin{cases} |e_{j,k}^H|, \text{ if } e_{j,k}^H := (v_j, v_k) \in E^H \\ 0, \text{ otherwise.} \end{cases}$$

In order to formulate the desired instance of the EMD, we take the point sets to be $P = \{u_i\}_{i=1}^{m+1}$ and $Q = \{v_j\}_{j=1}^{n+1}$. Here, u_{m+1} and v_{n+1} have been taken to be a dummy supplier and dummy consumer, respectively, to incorporate vertex deletion into our GMD framework. The weights on the sites are defined as follows:

$$w_{u_i} = 1 \text{ for } i = 1 \ldots, m \text{ and } w_{u_{m+1}} = n\ .$$

And,

$$w_{v_j} = 1 \text{ for } j = 1 \ldots, n \text{ and } w_{v_{m+1}} = m\ .$$

We note that the feasibility condition (2) is satisfied: $m+n$ is the total weight for both P and Q. An instance of the transportation problem is depicted in Fig. 3.

Finally, the ground distance from u_i to v_j is defined by:

$$d_{i,j} = \begin{cases} C_V|u_i - v_j| + & C_E\|E_i^G D_{m\times p} - E_j^H D_{n\times p}\|_1, \\ & \text{if } 1 \le i \le m, 1 \le j \le n \\ C_E\|E_j^H\|_1, & \text{if } i = m+1 \text{ and } 1 \le j \le n \\ C_E\|E_i^G\|_1, & \text{if } 1 \le i \le m \text{ and } j = n+1 \\ 0, & \text{otherwise.} \end{cases}$$

Here, $p = \min\{m, n\}$, the 1–norm of a row vector is denoted by $\|\cdot\|_1$, and D denotes a diagonal matrix with the all diagonal entries being 1.

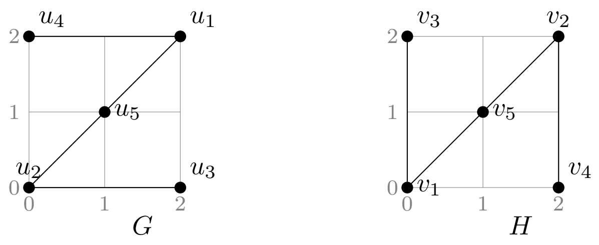


Figure 4: For the geometric graph $G, H \in \mathcal{G}^O(\mathbb{R}^2)$, the GMD is zero. The optimal flow is given by the matching $\pi(u_1) = v_2$, $\pi(u_2) = v_1$, $\pi(u_3) = v_4$, $\pi(u_4) = v_3$, and $\pi(u_5) = v_5$.

3.3 Metric Properties

We can see that the GMD induces a pseudo-metric on the space of ordered geometric graphs $\mathcal{G}^O(\mathbb{R}^d)$. Non-negativity, symmetry, and triangle inequality follow from those of the cost matrix $[d_{i,j}]$ defined in the GMD.

In addition, we note that $G = H$ (as ordered graphs) implies that $d_{i,j} = 0$ whenever $i = j$. The trivial flow, where each u_i sends its full supply to v_i, has a zero cost. So, $\text{GMD}(G, H) = 0$. The GMD does not, however, satisfy the separability condition on $\mathcal{G}^O(\mathbb{R}^d)$.

For the graphs G, H shown in Fig. 4, we have $\text{GMD}(G, H) = 0$. We note that G, H have the following adjacency length matrices $[E_i^G]_i$ and $[E_j^H]_j$, respectively:

$$\begin{bmatrix} 0 & 0 & 0 & 2 & \sqrt{2} \\ 0 & 0 & 2 & 0 & \sqrt{2} \\ 0 & 2 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ \sqrt{2} & \sqrt{2} & 0 & 0 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 & 2 & 0 & \sqrt{2} \\ 0 & 0 & 0 & 2 & \sqrt{2} \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ \sqrt{2} & \sqrt{2} & 0 & 0 & 0 \end{bmatrix}.$$

It can be easily checked that the flow that transports a unit of supply from $u_1 \mapsto v_2$, $u_2 \mapsto v_1$, $u_3 \mapsto v_4$, $u_4 \mapsto v_3$, $u_5 \mapsto v_5$, and five units from $u_6 \mapsto v_6$ has total cost zero. So, $\text{GMD}(G, H) = 0$. However, the graphs G and H are not the same geometric graph. The fact that $\text{GGD}(G, H) \ne 0$ implies the GGD is not stable under the GMD.

One can easily find even simpler configurations for two distinct geometric graphs with a zero GMD—if the graphs are allowed to have multiple connected components.

We conclude this section by stating a stability result for the GMD under the Hausdorff distance. We omit the proof, since it uses a similar argument presented in Theorem 1.

Theorem 2 (Hausdorff Stability of GMD) *Let $G, H \in \mathcal{G}^O(\mathbb{R}^d)$ be ordered geometric graphs with a bijection $\pi : V^G \to V^H$ such that $e^G_{i,j} = e^H_{\pi(i),\pi(j)}$ for all i, j. If $\delta > 0$ is such that $|u_i - \pi(u_i)| \le \delta$ for all $u_i \in V^G$, then*

$$\mathrm{GMD}(G, H) \le C_V |V^G| \delta.$$

3.4 Computing the GMD

As pointed out earlier, the GMD can be computed as an instance of transportation problem—using, for example, the network simplex algorithm. If the graphs have at most n vertices, computing the ground cost matrix $[d_{i,j}]$ takes $O(n^3)$-time. Since the bipartite network has $O(n)$ vertices and $O(n^2)$ edges, the simplex algorithm runs with a time complexity of $O(n^3)$, with a pretty good constant. Overall, the time complexity of the GMD is $O(n^3)$.

4 Experimental Results

We have implemented the GMD in Python, using network simplex algorithm from the `networkx` package. We ran a pattern retrieval experiment on letter drawings from the IAM Graph Database [18]. The repository provides an extensive collection of graphs, both geometric and labeled.

In particular, we performed our experiment on the `LETTER` database from the repository. The graphs in the database represent distorted letter drawings. The database considers only 15 uppercase letters from the English alphabet: `A`, `E`, `F`, `H`, `I`, `K`, `L`, `M`, `N`, `T`, `V`, `W`, `X`, `Y`, and `Z`. For each letter, a prototype line drawing has been manually constructed. On the prototypes, distortions are applied with three different level of strengths: `LOW`, `MED`, and `HIGH`, in order to produce 2250 letter graphs for each level. Each test letter drawing is a graph with straight-line edges; each node is labeled with its two-dimensional coordinates. Since some of the graphs in the dataset were not embedded, we had to compute the intersections of the intersecting edges and label them as nodes. The preprocessing guaranteed that all the considered graphs were geometric; a prototype and a distorted graph are shown in Fig. 5.

We devised a classifier for these letter drawings using the GMD. For this application, we chose $C_V = 4.5$ and $C_E = 1$ heuristically for the best results. For a test letter, we computed its GMD from the 15 prototypes, then sorted the prototypes in an increasing order of their distance to the test graph. We then check if the letter generating the test graph is among the first k prototypes. For each level of distortion and various values of k, we present the rate at which the correct letter has been found in the first k models. The summary of the empirical results have been shown in Table 1. Although

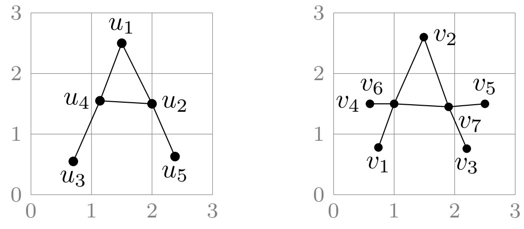


Figure 5: The prototype geometric graph of the letter `A` is shown on the left. On the right, a (`MED`) distorted letter `A` is shown.

	correct letter in first k models (%)		
Distortion	$k = 1$	$k = 3$	$k = 5$
`LOW`	96.66%	98.93%	99.37%
`MED`	66.66%	85.37%	91.15%
`HIGH`	73.73%	90.48%	95.51%

Table 1: Empirical result on the `LETTER` dataset

the graph edit distance based k-NN classifier still outperforms the GMD by a very small margin, our results has been extremely satisfactory.

One possible reason why the GMD might fail to correctly classify some of the graphs is that lacks the separability property as a metric.

5 Discussions

We have successfully introduced an efficiently computable and meaningful similarity measure for geometric graphs. However, the GMD lacks some of the desirable properties, like separability and stability. The currently presented stability results for the GGD and GMD have a factor that depends on the size of the input graphs. The question remains if the distance measures are in fact stable under much weaker conditions, possibly with constant factors on the right side. It will also be interesting to study the exact class of geometric graphs for which the GMD is, in fact, a metric.

References

[1] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. *Map Construction Algorithms.* Springer International Publishing, first edition, 2015.

[2] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Always learning. Pearson, 2013.

[3] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, May 1983.

[4] O. Cheong, J. Gudmundsson, H.-S. Kim, D. Schymura, and F. Stehn. Measuring the Similarity of Geometric

Graphs. In J. Vahrenhold, editor, *Experimental Algorithms*, volume 5526, pages 101–112. Springer, 2009.

[5] D. G. Corneil and C. C. Gotlieb. An efficient algorithm for graph isomorphism. *J. ACM*, 17(1):51–64, 1970.

[6] K.-S. Fu and B. Bhargava. Tree systems for syntactic pattern recognition. *IEEE Transactions on Computers*, C-22(12):1087–1099, 1973.

[7] K.-S. Fu and P. Swain. On syntactic pattern recognition. In J. T. Tou, editor, *Computer and Information Sciences – 1969*, volume 2 of *SEN Report Series Software Engineering*, pages 155–182. Elsevier, 1971.

[8] C. J. Hargreaves, M. S. Dyer, M. W. Gaultois, V. A. Kurlin, and M. J. Rosseinsky. The Earth Mover's Distance as a Metric for the Space of Inorganic Compositions. *Chemistry of Materials*, 32(24):10610–10620, Dec. 2020.

[9] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, Aug. 2006.

[10] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger. From Word Embeddings To Document Distances. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 957–966. PMLR, June 2015. ISSN: 1938-7228.

[11] J. Liu and Y. T. Lee. Graph-based method for face identification from a single 2d line drawing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1106–1119, 2001.

[12] J. Llados, E. Marti, and J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, 2001.

[13] S. Majhi and C. Wenk. Distance measures for geometric graphs. *arXiv preprint arXiv:2209.12869*, 2022.

[14] G. Monge. *Mémoire sur la théorie des déblais et des remblais.* Imprimerie royale, 1781.

[15] O. Pele and M. Werman. A Linear Time Histogram Metric for Improved SIFT Matching. In D. Forsyth, P. Torr, and A. Zisserman, editors, *Computer Vision – ECCV 2008*, Lecture Notes in Computer Science, pages 495–508, Berlin, Heidelberg, 2008. Springer.

[16] J. W. Raymond and P. Willett. Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2D chemical structure databases. *Journal of Computer-Aided Molecular Design*, 16(1):59–71, 2002.

[17] Z. Ren, J. Yuan, and Z. Zhang. Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 1093–1096, New York, NY, USA, Nov. 2011. Association for Computing Machinery.

[18] K. Riesen and H. Bunke. IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 5342, pages 287–297. Springer, 2008.

[19] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40(2):99–121, Nov. 2000.

[20] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362, May 1983.

[21] P. Willett. Similarity Searching in Databases of Three-Dimensional Chemical Structures. In H.-H. Bock, W. Lenski, and M. M. Richter, editors, *Information Systems and Data Analysis*, pages 280–293. Springer, 1994.

Metric and Path-Connectedness Properties of the Fréchet Distance for Paths and Graphs

Erin Chambers[*] Brittany Terese Fasy[†] Benjamin Holmgren [‡] Sushovan Majhi [§] Carola Wenk [¶]

Abstract

The Fréchet distance is often used to measure distances between paths, with applications in areas ranging from map matching to GPS trajectory analysis to handwriting recognition. More recently, the Fréchet distance has been generalized to a distance between two copies of the same graph embedded or immersed in a metric space; this more general setting opens up a wide range of more complex applications in graph analysis. In this paper, we initiate a study of some of the fundamental topological properties of spaces of paths and of graphs mapped to $\mathbb{R}^n$ under the Fréchet distance, in an effort to lay the theoretical groundwork for understanding how these distances can be used in practice. In particular, we prove whether or not these spaces, and the metric balls therein, are path-connected.

1 Introduction

One-dimensional data in a Euclidean ambient space is heavily studied in the computational geometry literature, and is central to applications in GPS trajectory and road network analysis [2,11,13,24]. One widely used distance measure on one-dimensional data is the Fréchet distance, which accounts for both geometric closeness as well as the connectivity of the paths or graphs being compared [1,4–8,10–14,16–18,20–22]. We build a theoretical foundation for these application areas by investigating spaces of paths and graphs in $\mathbb{R}^n$, including their metric and topological properties, under the Fréchet distance. The motivation for this work is simple: as practical approaches to compute the Fréchet distance between paths [6,14] and between graphs [10,18,20] grow in popularity, it is natural to inquire about the fundamental properties of such distances, in an effort to better understand exactly what they are capturing.

[*]Department of Computer Science, Saint Louis University, erin.chambers@slu.edu

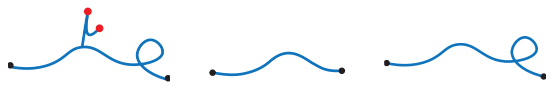
[†]School of Computing, Department of Mathematics, Montana State University brittany.fasy@montana.edu

[‡]School of Computing, Montana State University benjamin.holmgren1@student.montana.edu

[§]School of Information, University of California, Berkeley smajhi@ischool.berkeley.edu

[¶]Department of Computer Science, Tulane University cwenk@tulane.edu

We begin by defining the Fréchet distance between paths and graphs. Using open balls under the Fréchet distance to generate a topology, we study the metric and topological properties of the induced spaces. In particular, we work with three classes of paths: the space $\Pi_\mathcal{C}$ of all paths in $\mathbb{R}^n$, the space $\Pi_\mathcal{E}$ of all paths in $\mathbb{R}^n$ that are embeddings (i.e., maps that are homeomorphisms onto the image), and the space $\Pi_\mathcal{I}$ of all paths in $\mathbb{R}^n$ that are immersions (local embeddings). See Figure 1 for examples of paths in $\mathbb{R}^2$. In addition, we study the three analogous spaces of graphs: the sets $\mathcal{G}_\mathcal{C}$, $\mathcal{G}_\mathcal{I}$, and $\mathcal{G}_\mathcal{E}$ of continuous maps, immersions, and embeddings of graphs, respectively. This paper establishes the core metric and topological properties of the Fréchet distance on graphs and paths in Euclidean space.



(a) Continuous (b) Embedding (c) Immersion

Figure 1: Example of paths continuously mapped, embedded, and immersed in $\mathbb{R}^2$. The space of continuous maps allows arbitrary self-intersection on a path including backtracking (which occurs at the two red points); embeddings must induce homeomorphisms onto their image; and immersions are locally embeddings.

2 Background

In this section, we establish the definitions and notation from geometry and topology used throughout. We assume basic knowledge of concepts in topology. For common definitions central to this paper, we refer readers to Appendix A, or for greater detail, to [9,19].

Definition 1 (Types of Maps) *Let $\mathbb{X}$ and $\mathbb{Y}$ be topological spaces. A map $\alpha\colon \mathbb{X} \to \mathbb{Y}$ is called* continuous *if for each open set $U \subset \mathbb{Y}$, $\alpha^{-1}(U)$ is open in $\mathbb{X}$. We call α an* embedding *if α is injective. Equivalently, an* embedding *is a continuous map that is homeomorphic onto its image. If α is locally an embedding, then we say that α is an* immersion.

In particular, a continuous map $\gamma\colon [0,1] \to \mathbb{R}^n$ is called a *path* in $\mathbb{R}^n$. We call a path $\gamma\colon [0,1] \to \mathbb{R}^n$

rectifiable if γ has finite *length* (see Definition 32 in Appendix A.3). Moreover, we call a graph G rectifiable if there exists a finite cover of G such that every element in the cover is a rectifiable path.

Paths in $\mathbb{R}^n$ Letting $\widetilde{\Pi_{\mathcal{C}}}$ denote the set of all rectifiable paths in $\mathbb{R}^n$, we now define the path Fréchet distance.

Definition 2 (The Path Fréchet Distance [4]) *The* Fréchet distance $d_{FP}\colon \widetilde{\Pi_{\mathcal{C}}} \times \widetilde{\Pi_{\mathcal{C}}} \to \bar{\mathbb{R}}_{\geq 0}$ *between $\gamma_1, \gamma_2 \in \widetilde{\Pi_{\mathcal{C}}}$ is defined as:*

$$d_{FP}(\gamma_1, \gamma_2) := \inf_{r\colon [0,1]\to[0,1]} \max_{t\in[0,1]} ||\gamma_1(t) - \gamma_2(r(t))||_2,$$

where r ranges over all homeomorphisms such that $r(0) = 0$, and $||\cdot||_2$ denotes the Euclidean norm.

Graphs Mapped to $\mathbb{R}^n$ We define a *graph* $G = (V, E)$ as a finite set of vertices V and a finite set of edges E. Self-loops and multiple edges between a pair of vertices are allowed.[1] We topologize a graph by thinking of it as a CW complex; see Appendix A.1. If $\phi\colon G \to \mathbb{R}^d$ is a map, then we call (G, ϕ) a graph-map pair. We extend the path Fréchet distance to the Fréchet distance between graphs continuously mapped into $\mathbb{R}^n$:

Definition 3 (Graph Fréchet Distance) *Let $(G, \phi), (H, \psi)$ be continuous, rectifiable graph-map pairs. We define the Fréchet distance between (G, ϕ) and (H, ψ) by minimizing over all homeomorphisms:*[2]

$$d_{FG}\left((G,\phi),(H,\psi)\right) := \begin{cases} \inf_h ||\phi - \psi\circ h||_\infty & G \cong H. \\ \infty & \text{otherwise.} \end{cases}$$

For simplicity of exposition, when $G \cong H$, we write the LHS of this equation as $d_{FG}(\phi, \psi)$. Furthermore, defining the infimum over an emptyset to be ∞, the graph Fréchet distance is given by the following equation:

$$d_{FG}\left((G,\phi),(H,\psi)\right) := \inf_h ||\phi - \psi\circ h||_\infty,$$

where the infimum is taken over all homeomorphisms $h\colon G \to H$.

Note that if $G = H$ and ϕ is a reparameterization of ψ, then $d_{FG}(\phi, \psi) = 0$.

Observation 1 (Paths as Graphs) *If $G = [0,1]$ and $\alpha, \beta\colon [0,1] \to \mathbb{R}^n$ are paths, then the relationship between path and graph Fréchet distances is as follows:*

$$d_{FG}\left(\alpha,\beta\right) = \min\left\{d_{FP}(\alpha,\beta), d_{FP}(\alpha,\beta^{-1})\right\},$$

where $\beta^{-1}\colon I \to \mathbb{R}^n$ is defined by $\beta^{-1}(t) = \beta(1-t)$.

[1]Some references would call this a *multi-graph*, but for simplicity, we just use the term *graph*.

[2]Other generalizations of the Fréchet distance minimize over all "orientation-preserving" homeomorphisms, which can be defined in several ways for stratified spaces, and sometimes adding an orientation is not natural. Thus, we drop this requirement in our definition.

3 Metric Properties

We now address the question: Is this distance a metric? If not, can it be metrized? A well-known known property of the path Fréchet distance is that it is a pseudo-metric [4, 21]. That is, it satisfies all metric properties except for separability. We proof this property for d_{FG}.

Theorem 4 (Metric Properties of d_{FG}) *d_{FG} is an extended pseudo-metric that does not satisfy separability. When restricted to a homeomorphism class of graphs, d_{FG} is a pseudo-metric.*

Proof. We first prove that d_{FG} is an extended pseudo-metric (see Definition 27 in Appendix A.3).

Identity: Taking h to be the identity map in Definition 3, we find $d_{FG}((G,\phi_1),(G,\phi_1)) = 0$.

Symmetry: Consider $d_{FG}(\phi_1, \phi_2)$. If $G \ncong H$, then no homeomorphism $h\colon G \to H$ exists. Likewise, no homeomorphism $h'\colon H \to G$ exists. And, so,

$$d_{FG}((G,\phi_1),(H,\phi_2))) = \infty = d_{FG}((H,\phi_2),(G,\phi_1))).$$

Otherwise, since h is a homeomorphism, it is invertible. Thus, we can rewrite this as:

$$d_{FG}(\phi_1,\phi_2) = \inf_{h^{-1}} ||\phi_1\circ h^{-1} - \phi_2||_\infty = d_{FG}(\phi_2,\phi_1).$$

Subadditivity (the triangle inequality): Consider $d_{FG}((G_1,\phi_1),(G_2,\phi_2)) + d_{FG}((G_2,\phi_2),(G_3,\phi_3))$. If $G_1 \ncong G_2$, then $d_{FG}((G_1,\phi_1),(G_2,\phi_2)) = \infty$, and we are done. A symmetric argument follows for $G_2 \ncong G_3$. Thus, we assume $G_1 \cong G_2 \cong G_3$. Using the definition of Fréchet distance and the fact that the infimum is taken over homeomorphisms, we obtain:

$$\begin{aligned} d_{FG}(\phi_1,\phi_2) &+ d_{FG}(\phi_2,\phi_3) \\ &= \inf_{h'} ||\phi_1 - \phi_2\circ h'||_\infty + \inf_{h''} ||\phi_2 - \phi_3\circ h''||_\infty. \\ &\geq \inf_{h'} ||\phi_1 - \phi_2\circ h'||_\infty + \inf_{h,h'} ||\phi_2\circ h' - \phi_3\circ h||_\infty \\ &= \inf_{h,h'} ||\phi_1 + (\phi_2\circ h' - \phi_2\circ h') - \phi_3\circ h||_\infty \\ &= \inf_h ||\phi_1 - \phi_3\circ h||_\infty \\ &= d_{FG}(\phi_1,\phi_3). \end{aligned}$$

And so, we conclude that d_{FG} satisfies subadditivity.

Noting that if $G \ncong H$ that $d_{FG}\left((G,\phi),(H,\psi)\right) = \infty$, we conclude that d_{FG} is an extended pseudo-metric. However, the graph Fréchet distance between homeomorphic graphs is at most the Hausdorff distance between the images of the two maps. Thus, when restricted to a homeomorphism class of graphs, d_{FG} is a pseudo-metric. □

The only metric property not satisfied is separability. In order to metrize this pseudo-metric, we define $\mathcal{G}_{\mathcal{C}}(G)$ to be the the set of equivalence classes of continuous, rectifiable maps $G \to \mathbb{R}^n$, where two maps, ϕ_1

and ϕ_2, are equivalent if and only if $d_{FG}(\phi_1, \phi_2) = 0$. We write $[\phi_i]$ to denote the equivalence class of maps containing ϕ_i. We define two subspaces of $\mathcal{G}_\mathcal{C}(G)$: those representing immersions and embeddings, denoted $\mathcal{G}_\mathcal{I}(G)$ and $\mathcal{G}_\mathcal{E}(G)$, respectively. Note that $\mathcal{G}_\mathcal{E}(G) \subsetneq \mathcal{G}_\mathcal{I}(G) \subsetneq \mathcal{G}_\mathcal{C}(G)$. Let $\mathcal{G}_\mathcal{C}$ denote the induced set of equivalence classes of all graph-map pairs $(G, [\phi])$ such that $[\phi] \in \mathcal{G}_\mathcal{C}(G)$. Similiarly, we define $\mathcal{G}_\mathcal{I}$ and $\mathcal{G}_\mathcal{E}$, and note $\mathcal{G}_\mathcal{E} \subsetneq \mathcal{G}_\mathcal{I} \subsetneq \mathcal{G}_\mathcal{C}$. Hence,

Corollary 5 (Metric Extension for Graphs) *For every graph G, the graph Fréchet distance is a metric on the quotient spaces $\mathcal{G}_\mathcal{C}(G)$, $\mathcal{G}_\mathcal{I}(G)$, and $\mathcal{G}_\mathcal{E}(G)$. Moreover, the graph Fréchet distance is an extended metric on $\mathcal{G}_\mathcal{C}$, $\mathcal{G}_\mathcal{I}$, and $\mathcal{G}_\mathcal{E}$.*

Similarly, we consider paths in $\mathbb{R}^n$: in particular, $\Pi_\mathcal{C}$ is the set of equivalences classes of $\widetilde{\Pi_\mathcal{C}}$ up to orientation-preserving reparameterization. Equivalently, for $\gamma_1, \gamma_2 \in \widetilde{\Pi_\mathcal{C}}$, γ_1 is equivalent to γ_2 iff $d_{FP}(\gamma_1, \gamma_2) = 0$. Likewise, $\Pi_\mathcal{E}$ and $\Pi_\mathcal{I}$ are the subspaces of embeddings and immersions. Note that $\Pi_\mathcal{E} \subsetneq \Pi_\mathcal{I} \subsetneq \Pi_\mathcal{C}$. We topologize these spaces using the open ball topology (Appendix A.3). Again, by construction, we obtain:

Corollary 6 (Metric Properties of d_{FP}) *The path Fréchet distance is a metric on $\Pi_\mathcal{C}, \Pi_\mathcal{I}$ and $\Pi_\mathcal{E}$.*

4 Path-Connectedness Property

We now examine path-connectedness properties. See Definition 30 and Definition 31 of Appendix A.4 for definitions of path-connectivity.

4.1 Continuous Mappings

We start with the most general spaces of paths and graphs: the continuous, rectifiable maps into $\mathbb{R}^n$. In Euclidean spaces, linear interpolation is a useful tool because it defines the shortest paths between two points. In function spaces, linear interpolation is also nice:

Definition 7 (Linear Interpolation) *Let G be a graph and $\phi_0, \phi_1 \colon G \to \mathbb{R}^n$ be continuous, rectifiable maps. The* linear interpolation *from ϕ_0 to ϕ_1 is the map $\Gamma \colon [0,1] \to \mathcal{G}_\mathcal{C}(G)$ sending $t \in [0,1]$ to (G, ϕ_t), where:*

$$\phi_t := (1-t)\phi_0 + t(\phi_1 \circ h_*). \tag{1}$$

For ease of notation, we sometimes write $\Gamma_t := \Gamma(t)$.

Note that $(1-t)\phi_0 + t\phi_1$ is a linear combination of ϕ_0 and ϕ_1 (using $c_0 = 1-t$ and $c_1 = t$ in Definition 34). Thus, Γ is a continuous family of linear combinations of the maps ϕ_0 and ϕ_1; we show Γ is continuous in Lemma 35. in Appendix B.1. If $G = [0,1]$, the linear interpolation between graphs is simply linear interpolation between paths. For an example of linear interpolation between graphs, see Figure 4 in Appendix B.1.

However, linear interpolation is not well-defined in $\mathcal{G}_\mathcal{C}$, as we could have $\phi_1, \phi_2 \in [\phi] \in \mathcal{G}_\mathcal{C}(G)$. In fact, $\Gamma(t; \phi_1, \phi_2) = \Gamma(t; \phi_1, \phi_3)$ if and only if $\phi_1 = \phi_2$.

Definition 8 (Family of Interpolations) *Let G be a graph and $[\phi_0], [\phi_1] \in \mathcal{G}_\mathcal{C}(G)$. We define $\mathcal{C}([\phi_0], [\phi_1])$ to be the set of all linear interpolations between elements of $[\phi_0]$ and of $[\phi_1]$.*

We now demonstrate the existence of a family of interpolations between any two equivalence classes within $(\mathcal{G}_\mathcal{C}(G), d_{FG})$, proving path-connectivity.

Theorem 9 (Continuous Maps of Graphs) *For every graph G, the extended metric space $(\mathcal{G}_\mathcal{C}(G), d_{FG})$ is path-connected. Moreover, the connected components of $(\mathcal{G}_\mathcal{C}, d_{FG})$ are in one-to-one correspondence with the homeomorphism classes of graphs.*

Proof. Let $[\phi_0], [\phi_1] \in \mathcal{G}_\mathcal{C}(G)$. Let $\Gamma \in \mathcal{C}([\phi_0], [\phi_1])$. By Lemma 35 in Appendix B.1, Γ is continuous, and so $(\mathcal{G}_\mathcal{C}(G), d_{FG})$ is path-connected.

Moreover, suppose $(G, [\phi_0]), (H, [\phi_1]) \in \mathcal{G}_\mathcal{C}$ for the graphs G, H which are not homeomorphic. Then, $d_{FG}((G, [\phi_0]), (H, [\phi_1])) = \infty$, and connected components of the extended metric space $\mathcal{G}_\mathcal{C}$ are vacuously in one-to-one correspondence with homeomorphism classes of graphs. $\square$

Setting $G = [0,1]$, an identical proof holds for paths.

Corollary 10 (Continuous Maps of Paths) *The space $\Pi_\mathcal{C}$ is path-connected.*

We now demonstrate the stricter property of the path-connectivity of open distance balls:

Lemma 11 (Metric Balls in $(\mathcal{G}_\mathcal{C}, d_{FP})$) *Metric balls with finite radius in $(\mathcal{G}_\mathcal{C}, d_{FP})$ are path-connected.*

Proof. Let $\delta \in \mathbb{R}$ such that $\delta > 0$. Let $(G, [\phi_0]) \in \mathcal{G}_\mathcal{C}$.

Consider the metric ball $\mathbb{B} := \mathbb{B}_{d_{FG}}([\phi_0], \delta)$ in $\mathcal{G}_\mathcal{C}$. Let $[\phi_1], [\phi_2] \in \mathbb{B}$. We wish to find a path from $[\phi_1]$ to $[\phi_2]$. We first find a path in $\mathbb{B}_{d_{FG}}([\phi_0], \delta)$ from $[\phi_0]$ to $[\phi_2]$, as follows. Set

$$\varepsilon = \delta - d_{FG}([\phi_0], [\phi_2]).$$

By Lemma 25, we know that there exists a homeomorphism $h_* : G \to G$ such that the following inequality holds: $||\phi_0 - \phi_2 \circ h_*||_\infty < d_{FG}([\phi_0], [\phi_2]) + \varepsilon/2$.

Let $\Gamma \in \mathcal{C}([\phi_0],[\phi_2])$. Then, for all $t \in (0,1)$,

$$\begin{aligned} &d_{FG}(\Gamma_t, \phi_0) \\ &\quad = \inf_h ||((1-t)\phi_0 + t(\phi_2 \circ h_*)) - \phi_0 \circ h||_\infty \\ &\quad \leq ||((1-t)\phi_0 + t(\phi_2 \circ h_*)) - \phi_0 \circ h_*||_\infty \\ &\quad < d_{FG}([\phi_0],[\phi_2]) + \varepsilon/2 \\ &\quad < \delta. \end{aligned}$$

Thus, $\Gamma_t \in \mathbb{B}_{d_{FG}}([\phi_1],\delta)$, which means there exists a path from ϕ_0 to ϕ_2. Similarly, we find a path Γ' from ϕ_1 to ϕ_0. Concatentating the two paths, $\Gamma' \# \Gamma$ we have a path in $\mathbb{B}_{d_{FG}}([\phi_0],\delta)$ from $[\phi_1]$ to $[\phi_2]$. Hence, metric balls with finite radius in $\mathcal{G}_\mathcal{C}$ are path-connected. □

Setting $G = [0,1]$, we obtain:

Corollary 12 (Metric Balls in $(\Pi_\mathcal{C}, d_{FP})$) *Balls in the extended metric space $(\Pi_\mathcal{C}, d_{FP})$ are path-connected.*

4.2 Immersions

An immersion is a map that is locally injectivite. Thus, self-intersections are allowed, but a map pausing or backtracking is not. Next, we define these notions, and give examples in Figure 2.

Definition 13 (Pausing) *We say that a path γ* pauses *in an interval $I \subset [0,1]$ if $\gamma(x) = \gamma(y)$ for every $x, y \in I$. In this case, $[\gamma] \notin \Pi_\mathcal{I}$.*

Another possible violation of local injectivity is *backtracking* on a path.

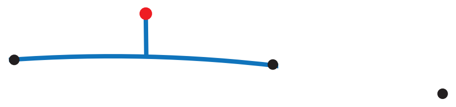
Definition 14 (Backtracking) *We say that a path γ is* backtracking *at a point $x \in [0,1]$ if there exists $\delta > 0$ such that for every $\epsilon \in (0,\delta)$, either $\gamma|_{(x-\epsilon,x)} \subset \gamma_{(x,x+\epsilon)}$ or $\gamma|_{(x,x+\epsilon)} \subset \gamma|_{(x-\epsilon,x)}$.*

To show the path-connectivity of spaces of immersions, the proof in Theorem 9 for continuous mappings is *almost* sufficient, but these added violations must be addressed. Thus, we introduce additional maneuvers to avoid pauses and backtracking.

Lemma 15 (Rerouting Pauses) *Let $\gamma_0, \gamma_1 \in \widetilde{\Pi_\mathcal{I}}$, and let $\Gamma : [0,1] \to \widetilde{\Pi_\mathcal{C}}$ be a path in $\widetilde{\Pi_\mathcal{C}}$ from γ_0 to γ_1. Suppose there exists an interval $[t_1,t_2]$ such that for all $t \in [0,1] \setminus (t_1,t_2)$, Γ_t is an immersion. But, for all $t \in (0,1)$, Γ_t has a single pause. Then, there exists a different path $\Gamma^* : [0,1] \to \Pi_\mathcal{I}$ that avoids the pause.*

Proof. Let $t \in [t_1,t_2]$. Let the pause in Γ_t be over the interval $(a_t,b_t) \subset [0,1]$. Let $\varepsilon_t := \min(t_2-t, t-t_1)$. We stretch the paused interval (a_t,b_t) in Γ_t by defining a map $\Gamma^*_t : [0,1] \to \widetilde{\Pi_\mathcal{I}}$ as follows:

- $\Gamma^*_t(-\infty, a_t]$ is an oriented reparameterization of $\Gamma_t(-\infty, a_t - \varepsilon_t]$.



(a) Forced Backtracking (b) Constant Map

Figure 2: Examples of paths in $\Pi_\mathcal{C}$ but not $\Pi_\mathcal{I}$. Figure 2a demonstrates a path with necessary backtracking at the red point. Figure 2b demonstrates a constant path which (vacuously) must pause. For a nontrivial example of a path with pauses, consider any parameterization of a path sending an open interval to a point.

- $\Gamma^*_t(a_t,b_t)$ is an oriented reparameterization of $\Gamma_t(a_t-\varepsilon_t, a_t] \# \Gamma_t[b_t, b_t+\varepsilon)$
- $\Gamma^*_t[b_t,\infty)$ is an oriented reparameterization of $\Gamma_t[b_t+\varepsilon_t,\infty)]$.

By construction, Γ^*_t has removed the pause between a_t and b_t; hence, $\Gamma_s \in \widetilde{\Pi_\mathcal{I}}$. Putting these maps together, we obtain a map $\Gamma^* : [0,1] \to \widetilde{\Pi_\mathcal{C}}$, where

$$\Gamma(t) := \begin{cases} \Gamma_t & \text{if } t \notin (t_1,t_2) \\ \Gamma^*_t & \text{if } t \in (t_1,t_2). \end{cases} \tag{2}$$

Moreover, Γ is continuous in $\Pi_\mathcal{I}$. □

Direct linear interpolation can also yield degeneracies by creating a singleton in specific circumstances, or by creating a backtracking point. Each are addressed in the following theorem, and a path is constructed.

Theorem 16 (Path Immersions) *The extended metric space $(\Pi_\mathcal{I}, d_{FP})$ of paths immersed in $\mathbb{R}^n$ is path-connected iff $n > 1$.*

Proof. If $n = 1$, it is easy to see that $\Pi_\mathcal{I}$ is not path-connected by examining intervals with reversed orientation which trivially degenerate to a point when constructing a path, violating local injectivity.

Now, consider $n > 1$. Let $[\gamma_0],[\gamma_1] \in \Pi_\mathcal{I}$. Using Definition 7, let $\Gamma : [0,1] \to \Pi_\mathcal{C}$ be the linear interpolation from γ_0 to γ_1. This interpolation is in $\Pi_\mathcal{C}$, not $\Pi_\mathcal{I}$, so we explain how to edit Γ so that it stays in $\Pi_\mathcal{I}$. If $\Gamma(t) \in \widetilde{\Pi_\mathcal{I}}$ for each $t \in [0,1]$, we are done. Otherwise, let $T \subset I$ be the set of times that introduce a non-immersion (i.e., $t \in T$ iff $\Gamma(t) \notin \widetilde{\Pi_\mathcal{I}}$, but $\Gamma(t-\epsilon) \in \widetilde{\Pi_\mathcal{I}}$ for all ϵ small enough). There are two things that might have happened at t: either an interval collapsed to a point (a pause) or backtracking was introduced in $\Gamma(t)$.

1. Suppose there exists $t \in T$ where an interval pauses as in Definition 13 and Figure 2b. Note that a pausing event occurs either if an interval of Γ_t becomes degenerate, or Γ_t collapses to a point.

If pausing occurs only on an open interval $(a, b) \subset [0, 1]$ of $\gamma_t \in \Gamma_t$, it can be avoided using Lemma 15. If pausing occurs on a closed interval $[a, b] \subset [0, 1]$, we convert it to the open set $(a-\varepsilon, b+\varepsilon)$ for small ε, and use Lemma 15. If either $a = 0$ or $b = 1$, we simply redefine γ_t to start at b or to end at a, respectively, using Lemma 37. The pausing event is guaranteed to conclude at some $t + \delta$ for $\delta \geq 0$ since $[\gamma_1] \in \Pi_{\mathcal{I}}$, and Γ must attain $\gamma_1 \in [\gamma_1]$.

If a pausing event stems from a full collapse to a singleton (i.e. interpolation occurs between two colinear segments with reverse orientation, and consequently degenerate to a point), the collapse can be circumvented by rotating the path defining Γ_t, which is done in Lemma 38.

2. Alternatively, suppose there exists $t \in T$ which corresponds to backtracking at a point in a path Γ_t according to Definition 14 and Figure 2a. Here, Γ_t can remain in $\widetilde{\Pi_{\mathcal{I}}}$ by inflating a ball of radius ϵ for sufficiently small $\epsilon > 0$ about the backtracking point before it is created. This is included in Lemma 39, and shown in Figure 6b.

For all $t \in T$, the described moves can be used to subvert lapses in local injectivity along Γ. Hence, we construct a path Γ by interpolating from γ_0 to γ_1, and applying the required move at each $t \in T$ to handle pauses or backtracking. By the arbitrariness of Γ, we have given a class of continuous paths from any element $\gamma_0 \in [\gamma_0]$ to any $\gamma_1 \in [\gamma_1]$. □

Theorem 17 (Metric Balls in $(\Pi_{\mathcal{I}}, d_{FP})$) *If $n > 1$, then balls in the extended metric space $(\Pi_{\mathcal{I}}, d_{FP})$ are path-connected.*

Proof. Let $[\gamma_0], [\gamma_1] \in \Pi_{\mathcal{I}}$, and let $\delta > 0$. Let $\Gamma \in \Pi_{\mathcal{I}}$ be the map Γ in the proof of Theorem 16. By Lemma 11, linear interpolation does not increase the Fréchet distance. By design, avoiding singleton degeneracies by way of Lemma 38 also does not increase the Fréchet distance. Moreover, by construction, the map Γ^* of Lemma 15 preserves the Fréchet distance. The maneuver in Lemma 39 could potentially increase $d_{FP}(\Gamma_t, \gamma_1)$ at some time $t \in [0, 1]$, but in this case any critical backtracking points can be perturbed slightly in order to no longer define the $d_{FP}(\Gamma_t, \gamma_1)$. Hence, these moves need not result in $d_{FP}(\Gamma_t, \gamma_1) > \delta$, meaning that $\Gamma_t \in \mathbb{B}_{d_{FP}}(\phi_1, \delta)$, and balls in $\Pi_{\mathcal{I}}$ are path-connected. □

We use the same maneuvers from Theorem 16 in the context for graphs under d_{FG}.

Theorem 18 (Graph Immersions) *For every graph G, the extended metric space $(\mathcal{G}_{\mathcal{I}}(G), d_{FG})$ is path-connected. Connected components of the extended metric space $(\mathcal{G}_{\mathcal{I}}, d_{FG})$ are in one-to-one correspondence with the homeomorphism classes of graphs.*

Proof. We construct Γ identically to Theorem 16, but interpolation occurs among each edge of G in $\mathcal{G}_{\mathcal{I}}(G)$ rather than between individual segments. As in Theorem 16, local injectivity can only be violated by pauses and backtracking on edges, which are handled using Lemma 15, Lemma 38, and Lemma 39 on each edge. If $(G, [\phi_0]), (H, [\phi_1]) \in \mathcal{G}_{\mathcal{I}}$ for G, H which are not homeomorphic, then $d_{FG}((G, [\phi_0]), (H, [\phi_1])) = \infty$. □

Similarly, we can adopt Theorem 17 for each edge in a graph to show path-connectivity of balls in $\mathcal{G}_{\mathcal{I}}$.

Theorem 19 (Metric Balls in $(\mathcal{G}_{\mathcal{I}}(G), d_{FG})$) *For every graph G, the balls in the extended metric space $(\mathcal{G}_{\mathcal{I}}(G), d_{FG})$ are path-connected.*

Proof. Let $[\phi] \in \mathcal{G}_{\mathcal{I}}$, and let $\delta > 0$. Let $\mathbb{B}$ be the intersection $\mathbb{B}_{d_{FG}}(\phi, \delta) \cap \mathcal{G}_{\mathcal{I}}(G)$. Let $[\phi_0], [\phi_1] \in \mathbb{B}$. Construct the path $\Gamma : [0, 1] \to \mathbb{B}$ from $[\phi_0]$ to $[\phi_1]$ in the same way as Theorem 18. Just as in Theorem 17, linear interpolation and the moves in Lemma 38, Lemma 39, and Lemma 15 mandate that $\Gamma(t) \in \mathbb{B}$ for every $t \in (0, 1)$ identically to the path Fréchet distance. □

4.3 Embeddings

Lastly, we examine the path-connectedness property of the analogous spaces of embeddings.

Theorem 20 (Path Embeddings) *The extended metric space $(\Pi_{\mathcal{E}}, d_{FP})$ is path-connected in $\mathbb{R}^n$ if and only if $n > 1$.*

Proof. If $n = 1$, two paths with reverse orientations are not path-connected.

Now, let $n > 1$, and let $[\gamma_0], [\gamma_1] \in \Pi_{\mathcal{E}}$. By Alexander's trick,[3] there exists $s_0 \in [0, 1]$ such that $\gamma_0' := \gamma_0|_{[s,1]}$ and $s_1 \in [0, 1]$ such that $\gamma_1' := \gamma_1|_{[s_1,1]}$, where s_0 and s_1 are nearly straight. Let $\angle$ be the angle between the segments γ_0' and γ_1'. Let $S\colon [\frac{1}{4}, \frac{2}{4}] \to \Pi_{\mathcal{E}}$ be the map rotating γ_0' by $\angle$ to become parallel with γ_1'. Finally, let Γ be the interpolation from $\gamma_0' \circ S$ to γ_1'.

Define $P\colon [0, 1] \to \Pi_{\mathcal{E}}$ as the resulting composition:

$$P(t) = \begin{cases} \gamma_0 \mid_{[(1-t)s,1]}, & t \in [0, \frac{1}{4}] \\ S(t), & t \in [\frac{1}{4}, \frac{2}{4}] \\ \Gamma(t) & t \in [\frac{2}{4}, \frac{3}{4}] \\ \gamma_1 \mid_{[(1-t)s,1]}, & t \in [\frac{3}{4}, 1]. \end{cases}$$

The steps attaining γ_0' and γ_1', as nothing else than a restriction of γ_0 and γ_1, are continuous. Moreover, S is continuous as the rotation of γ_0', and Γ is continuous by Lemma 35. By the arbitrariness of the constructed path and γ_0, γ_1, there is a family of continuous paths for any $\gamma_0 \in [\gamma_0], \gamma_1 \in [\gamma_1]$, and $\Pi_{\mathcal{E}}$ is path-connected. □

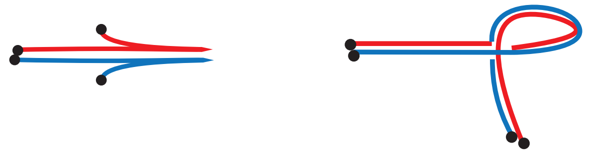


Figure 3: Two embedded paths γ_0, γ_1 in $\mathbb{R}^2$ and $\mathbb{R}^3$ respectively, for which constructing a path $\Gamma : [0,1] \to \Pi_{\mathcal{E}}, \Gamma(0) = \gamma_0, \Gamma(1) = \gamma_1$ is not possible without having $\Gamma(t) \notin \mathbb{B}_{d_{FP}}(\gamma_1, d_{FP}(\gamma_0, \gamma_1))$ for some $t \in [0,1]$.

Moreover, in high dimensions we can construct a path in $\Pi_{\mathcal{E}}$ not increasing the Fréchet distance.

Theorem 21 (Metric Balls in $(\Pi_{\mathcal{E}}, d_{FP})$) *If $n \geq 4$, then balls with finite radius in the extended metric space $(\Pi_{\mathcal{E}}, d_{FP})$ are path-connected in $\mathbb{R}^n$.*

Proof. If $n \geq 4$, the same map Γ given in Theorem 16 is sufficient, except that self-crossings must be avoided. At each $s \in [0,1]$ where a self-crossing would occur, we perturb Γ by a sufficiently small amount in order to avoid a self-crossing without increasing the Fréchet distance using the maneuver in Lemma 41. □

A simple examination shows that metric balls are not path-connected in low dimensions.

Theorem 22 (Metric Balls in $(\Pi_{\mathcal{E}}, d_{FP})$) *If $n \in \{1,2,3\}$, then balls with finite radius in the extended metric space $(\Pi_{\mathcal{E}}, d_{FP})$ are not path-connected in $\mathbb{R}^n$.*

Proof. For $n = 1$, let $[\gamma] \in \Pi_{\mathcal{E}}$. Let $\gamma^{-1} := \gamma(1-t)$, and note that $\gamma^{-1} \in \Pi_{\mathcal{E}}$. If $n = 2$, consider two paths within a fixed Fréchet ball that are much wider than their Fréchet distance. If $n = 3$, consider two paths with small Fréchet distance that form a loop, with one section passing under the other. If these loops have reversed orientation between the two paths, the Fréchet distance must increase. See Figure 3 for examples. □

In the setting for graphs, the path-connectedness property reduces to a knot theory problem if $n \leq 3$, and is not maintained. For $n \geq 4$, we use the existence of a sequence of Reidemeister moves [25] from any tame knot to another to construct paths in $\mathcal{G}_{\mathcal{E}}$.

Theorem 23 (Path-Connectivity of $(\mathcal{G}_{\mathcal{E}}, d_{FG})$, $n \geq 4$) *For all graphs G and $n \geq 4$, the extended metric space $(\mathcal{G}_{\mathcal{E}}(G), d_{FG})$ is path-connected. Moreover, connected components of the extended metric space $(\mathcal{G}_{\mathcal{E}}, d_{FG})$ are in one-to-one correspondence with homeomorphism classes of graphs.*

[3]Two embeddings of the n-ball are isotopic, first proven by Alexander [3]; see also [15, §4].

Proof. Let G be a graph, and $\phi_0, \phi_1 \in \mathcal{G}_{\mathcal{E}}(G)$. If $n \geq 4$, any tame knot can be unwound by a sequence of Reidemeister moves into the unknot. Construct $\Gamma : [0,1] \to \mathcal{G}_{\mathcal{E}}(G)$ by linear interpolating until some $t \in (0,1)$ causes Γ_t to self-intersect. At t, there exists a Reidemeister move allowing the crossing event to occur. Hence, any sequence of knots and free edges comprising ϕ_0 and ϕ_1 can be unwound to a sequence of unknots and straight edges, and then interpolated accordingly. Consequently, there exists a path from ϕ_0 to ϕ_1 in $(\mathcal{G}_{\mathcal{E}}(G), d_{FG})$. Note that we require ϕ_0, ϕ_1 are rectifiable in Section 2. □

In dimension 4 or higher, the path-connectivity of balls in $\mathcal{G}_{\mathcal{E}}(G)$ is shown in the same way as for paths.

Theorem 24 (Metric Balls in $(\mathcal{G}_{\mathcal{E}}(G), d_{FG})$) *For all graphs G and $n \geq 4$, metric balls in the space $(\mathcal{G}_{\mathcal{E}}(G), d_{FG})$ are path-connected.*

Proof. The proof is identical to that in Lemma 41, but Reidemeister moves are used for each edge in a graph rather than a single segment. □

5 Conclusion

In this paper, we studied some fundamental topological properties of spaces of paths and graphs in Euclidean space under the Fréchet distance. In particular, we investigated metric properties of the Fréchet distance on paths and graphs, as well as studying the path-connectedness of metric balls in the space of such graphs. While this work is theoretical and mathematical in nature, we feel that establishing the underlying properties of the topological spaces it can define provides an important theoretical backdrop, which is especially critical due to the widespread popularity of the Fréchet distance in computational geometry, and the growing popularity of its extension for graphs. Our contribution begins a careful study of the Fréchet distance and its topological properties. Extensions to this work abound, and include examining core topological properties of other distance measures in computational geometry, as well as other important properties of the Fréchet distance.

Acknowledgements

We thank our coauthors from [9] for the initial conversations that inspired this paper. In addition, the authors thank the National Science Foundation; specifically, Erin Chambers is partially supported by NSF awards 2106672 and 1907612, Brittany Terese Fasy and Benjamin Holmgren are partially supported by award 1664858, Fasy is also supported by award 2046730, and Carola Wenk is partially supported by award 2107434.

References

[1] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014.

[2] M. Ahmed and C. Wenk. Constructing street networks from gps trajectories. In *Algorithms – ESA 2012*, pages 60–71, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[3] J. W. Alexander. On the deformation of an n cell. *Proceedings of the National Academy of Sciences*, 9(12):406–407, 1923.

[4] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *IJCGA*, 5(1–2):75–91, 1995.

[5] H. Alt, C. Knauer, and C. Wenk. Matching polygonal curves with respect to the Fréchet distance. In A. Ferreira and H. Reichel, editors, *STACS 2001*, pages 63–74, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[6] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. Fréchet distance for curves, revisited. In Y. Azar and T. Erlebach, editors, *Algorithms – ESA 2006*, pages 52–63, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[7] K. Buchin, M. Buchin, and A. Schulz. Fréchet distance of surfaces: Some simple hard cases. In *European Symposium on Algorithms*, pages 63–74. Springer, 2010.

[8] K. Buchin, T. Ophelders, and B. Speckmann. Computing the Fréchet distance between real-valued surfaces. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2443–2455. ACM, 2017.

[9] M. Buchin, E. Chambers, P. Fang, B. T. Fasy, E. Gasparovic, E. Munch, and C. Wenk. Distances between immersed graphs: Metric properties. *La Matematica*, 2(1):197–222, 2023.

[10] M. Buchin, A. Krivosija, and A. Neuhaus. Computing the Fréchet distance of trees and graphs of bounded tree width. In *Proceedings of the 36th European Workshop on Computational Geometry*, 2020.

[11] E. Chambers, B. T. Fasy, Y. Wang, and C. Wenk. Map-matching using shortest paths. In *ACM Transactions on Spatial Algorithms and Systems*, pages 1–17. Association for Computing Machinery, 2020.

[12] E. W. Chambers, E. Colin de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010. Special Issue on 24th Annual Symposium on Computational Geometry (SoCG'08).

[13] D. Chen, A. Driemel, L. J. Guibas, A. Nguyen, and C. Wenk. Approximate map matching with respect to the frechet distance. pages 75–83, 2011.

[14] C. Colombe and K. Fox. Approximating the (continuous) Fréchet distance. In K. Buchin and E. Colin de Verdiére, editors, *37th International Symposium on Computational Geometry (SoCG, 2021)*, 2021.

[15] E. Denne and J. M. Sullivan. Convergence and isotopy type for graphs of finite total curvature. *Discrete differential geometry*, pages 163–174, 2008.

[16] A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

[17] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete and Computational Geometry*, 48(1):94–127, Feb. 2012.

[18] A. Driemel, I. van der Hoog, and E. Rotenburg. On the discrete fréchet distance in a graph. In M. Kerber and X. Goaoc, editors, *Proceedings of the Symposium on Computational Geometry, 2022.*

[19] H. Edelsbrunner and J. L. Harer. *Computational Topology: An Introduction.* American Mathematical Society, 2010.

[20] P. Fang and C. Wenk. The Fréchet distance for plane graphs. In *Proceedings of the 37th European Workshop on Computational Geometry*, 2021.

[21] M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Mathematico di Palermo*, 22:1–74, 1906. Paragraphs 78–80.

[22] A. F. C. IV and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Trans. Algorithms*, 7(1), dec 2010.

[23] L. Jiří. *Basic Analysis: Introduction to Real Analysis*, volume 5.6. 2022.

[24] M. Musleh, S. Abbar, R. Stanojevic, and M. Mokbel. Qarta: an ml-based system for accurate map services. *Proceedings of the VLDB Endowment*, 14(11):2273—2282, July 2021.

[25] K. Reidemeister. Knoten und gruppen. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 5, pages 7–23. Springer, 1927.

A Distances and Topology

Let $\bar{\mathbb{R}}$ denote the extended real line: $\bar{\mathbb{R}} = \mathbb{R} \cup \pm\infty$. We now provide the basic definitions relating to distances and topology used throughout this paper.

A.1 Graphs

Graphs are a central object studied in this paper.

Throughout this paper, we use the term *graph* to mean a *multi-graph*. A multi-graph $G = (V, E)$ is a finite set of vertices V and edges E. Self-loops and multiple edges between two vertices are allowed in this setting. A graph is an example of a more general structure called a CW complex, which we topologize as follows: (1) the topology on G restricted to V is the discrete topology; (2) for a edge e, the open sets restricted to is closure $\bar{e}$ are those induced by the subspace topology on $[0,1] \subset \mathbb{R}$ and a homeomorphism $[0,1] \to \bar{e}$; (3) we take the quotient topology on $(\cup_{v \in V}) \cup (\cup_{e \in E} \bar{e})$.

A.2 Fréchet Distance

We defined the path and graph Fréchet distances in Section 2. The path Fréchet distance is well-studied [1, 4–8, 10–14, 16–18, 20–22]. The graph Fréchet distance has been less studied, but many results for paths transfer to graphs.

The proof of the following lemma follows from the definition of Fréchet distance and the definition of infimum.

Lemma 25 (Approximator) *For all graphs G, if $[\phi_0], [\phi_1] \in \Pi_{\mathcal{C}}(G)$, then for every $\varepsilon > 0$, there exists a homeomorphism $h_* \colon G \to G$ such that*

$$||\phi_0 - \phi_1 \circ h||_\infty < d_{FG}(\phi_0, \phi_1) + \epsilon.$$

Proof. By Definition 3,

$$d_{FG}([\phi_1], [\phi_2]) = \inf_h ||\phi_1 - \phi_2 \circ h||_\infty.$$

Then, by the definition of infimum, for every $\varepsilon > 0$, there exists $h_* : G \to G$ such that

$$\begin{aligned} ||\phi_1 - \phi_2 \circ h_*||_\infty &< \inf_h ||\phi_1 - \phi_2 \circ h||_\infty + \varepsilon/2 \\ &= d_{FG}([\phi_1], [\phi_2]) + \varepsilon/2, \end{aligned}$$

as was to be shown. □

A.3 Defining Spaces from Distances and Metrics

Given a set $\mathbb{X}$ and a $d \colon \mathbb{X} \times \mathbb{X} \to \bar{\mathbb{R}}_{\geq 0}$, we topologize $\mathbb{X}$ as follows:

Definition 26 (The Open Ball Topology) *Let $\mathbb{X}$ be a set and $d \colon \mathbb{X} \times \mathbb{X} \to \bar{\mathbb{R}}_{\geq 0}$ a distance function. For each $r > 0$ and $x \in \mathbb{X}$,* *let $\mathbb{B}_d(x,r) := \{y \in \mathbb{X} \mid d(x,y) < r\}$. The open ball topology on $\mathbb{X}$ with respect to d is the topology generated by $\{\mathbb{B}_d(x,r) \mid x \in \mathbb{X}, r > 0\}$. We call $(\mathbb{X}, d)$ a* distance space*.*

In words, $\mathbb{B}_d(x,r)$ denotes the open ball of radius r centered at x with respect to d. We use these open balls to generate a topology on $\mathbb{X}$, allowing x to range over $\mathbb{X}$ and r to range over all positive real numbers.

We are particularly interested in distance functions that are either a pseudo-metric or a metric. These are defined as follows.

Definition 27 (Pseudo-Metric) *Let $\mathbb{X}$ be a set and let $d\colon \mathbb{X}\times\mathbb{X} \to \bar{\mathbb{R}}_{\geq 0}$ be a distance function. We call d a* pseudo-metric *on $\mathbb{X}$ if d satisfies the following:*

- *Finiteness: $d(x,y) < \infty$ for all $x, y \in \mathbb{X}$.*
- *Identity: $d(x,x) = 0$ for all $x \in \mathbb{X}$.*
- *Symmetry: $d(x,y) = d(y,x)$ for all $x, y \in \mathbb{X}$.*
- *Subadditivity (the triangle inequality): $d(x,z) \leq d(x,y) + d(y,z)$ for all $x, y, z \in \mathbb{X}$*

If d satisfies everything except finiteness, then we call d an extended pseudo-metric*.*

In order to be a metric, d must fulfill stricter criteria:

Definition 28 (Metric) *Let $\mathbb{X}$ be a set and let the function $d\colon \mathbb{X}\times\mathbb{X} \to \bar{\mathbb{R}}_{\geq 0}$ be a pseudo-metric. We say that d is a* metric *if d also satisfies:*

- *Seperability: for any $x, y \in \mathbb{X}$, if $x \neq y$, then $d(x,y) > 0$.*

Often, if $(\mathbb{X}, d)$ is a pseudo-metric space, a standard procedure is to define an equivalence class for $x, y \in \mathbb{X}$ where $x \sim y$ if $d(x,y) = 0$. Then, the quotient space $\mathbb{X}/_{\sim}$ is a metric space.

Common examples of metrics on function spaces are those induced by L_p-norms. For example, let $(\mathbb{Y}, d_{\mathbb{Y}})$ be distance space, let $\mathbb{X}$ be any topological space, and let $f, g\colon \mathbb{X} \to \mathbb{Y}$. Then, the distance induced by the L_∞-norm between f and g is:

$$||f-g||_\infty = \max_{x\in\mathbb{X}} d_{\mathbb{Y}}(f(x), f(y)).$$

A.4 Paths and Maps

With the basic definitions from topology in hand, we are equipped to define a property of fundamental interest in topology: path-connectedness.

Definition 29 (Path) *A* path *in a topological space $\mathbb{X}$ between two elements $a, b \in \mathbb{X}$, is defined to be a continuous map $\gamma : [0,1] \to \mathbb{X}$ where $\gamma(0) = a$, and $\gamma(1) = b$.*

Given two paths $\gamma_1, \gamma_2\colon [0,1] \to \mathbb{X}$ such that $\gamma_1(1) = \gamma_2(0)$, we combine them by taking both at double-speed. This is called the concatenation of paths. In particular, $\gamma_1 \# \gamma_2\colon [0,1] \to \mathbb{R}^n$ is defined by:

$$\gamma_1\#\gamma_2(t) := \begin{cases} \gamma_1(2t) & t \in [0, 0.5]. \\ \gamma_2(2t-1) & \text{otherwise.} \end{cases}$$

Given one path $\gamma\colon [0,1] \to \mathbb{X}$ and an interval $[a,b] \subseteq [0,1]$, the restriction of γ to $[a,b]$ is also a path, given by:

$$\gamma|_{[a,b]}(t) := \gamma(a + t(b-a)).$$

With the definition of paths, we define a primary property of interest in this paper: path-connectivity.

Definition 30 (Path-Connectivity) *A topological space $\mathbb{X}$ is called* path-connected *if there exists a* path *between any two elements in $\mathbb{X}$.*

We also define the path-connectedness property specifically for distance balls:

Definition 31 (Path-Connectivity of Balls) *Let $(\mathbb{X}, d)$ be a topological space, let $x \in \mathbb{X}$. We say that the distance balls in $(\mathbb{X}, d)$ are* path-connected *if for every $x \in \mathbb{X}$ and $r \in \mathbb{R}_{\geq 0}$, the distance ball $\mathbb{B}_d(y, r)$ is path-connected.*

And, the *length* of a path in a distance space is given by:

Definition 32 (Length) *Let $(\mathbb{X}, d)$ be a distance space and let γ be a* path *in $(\mathbb{X}, d)$. Let $\mathcal{P}$ be the set of all finite subsets $P = \{t_i\}$ of $[0,1]$ such that such that $0 = t_0 < t_1 < \ldots < t_n = 1$. The* length *$L_d(\gamma)$ of γ is:*

$$L_d(\gamma) := \sup_{P\in\mathcal{P}} \sum_{i=1}^{n} d(\gamma(t_i), \gamma(x_{i-1})).$$

Additionally, it is often useful in our setting to reparameterize paths, both to define the Fréchet distance and to maintain properties such as injectivity in a map.

Definition 33 (Reparameterization) *Let $\mathbb{X}, \mathbb{Y}$ be a topological spaces, $\phi\colon \mathbb{X} \to \mathbb{Y}$, and $h\colon \mathbb{X} \to \mathbb{X}$ is a homeomorphism. Then, we call $\phi \circ h$ a reparameterization of ϕ. In the setting where $\mathbb{X} = [0,1]$ and $h(0) = 0$, we call $\phi \circ h$ an* orientation-preserving *reparameterization.*

B Omitted Details for Path-Connectivity

In this appendix, we provide additional context for the proofs of path-connectivity in Section 4.

B.1 Additional Details on Interpolation

Given two continuous maps of the same graph into $\mathbb{R}^n$, we define the interpolation between them. First, we need to define linear combinations of graphs (and paths).

Definition 34 (Linear Combination of Graphs) *Let G be a graph, let $\phi_1, \phi_2 : [0,1] \to \mathbb{R}^n$ be continuous, rectifiable maps, and $c_1, c_2 \in \mathbb{R}$. Then, the linear combination $\phi = c_1\phi_1 + c_2\phi_2$ is defined as follows: the map $\phi\colon G \to \mathbb{R}^n$ is defined by $\phi(x) := c_2\phi_1(x)+c_2\phi_2(x)$. In this case, we may also say (G, ϕ) is a linear combination of graph-map pairs (G, ϕ_1) and (G, ϕ_2).*

In this definition, we observe that ϕ is continuous (since ϕ_0 and ϕ_1 are continuous), which means that linear combinations are well-defined in the set of all continuous, rectifiable maps. It is not well defined in the space $\mathcal{G}_\mathcal{C}(G)$ overall.

Lemma 35 (Linear Interpolation is Continuous) *For all graphs G, linear interpolation between graphs in $\mathcal{G}_\mathcal{C}(G)$ (and hence between homeomorphic graphs in $\mathcal{G}_\mathcal{C}$) is a continuous function.*

Proof. Let $[\phi_0], [\phi_1] \in \mathcal{G}_\mathcal{C}(G)$. Let $\Gamma\colon [0,1] \to \mathcal{G}_\mathcal{C}(G)$ be the linear interpolation from ϕ_0 to ϕ_1.

We prove that Γ satisfies the ε-δ definition of continuity. Let $\varepsilon > 0$. Set $\delta = \frac{\varepsilon}{d_{FG}([\phi_0],[\phi_1])}$. Let $s, t \in [0,1]$ such that $|s - t| < \delta$. Then, we have

$$\begin{aligned} &d_{FG}([\Gamma_t], [\Gamma_s]) \\ &\quad= d_{FG}([(1-t)\phi_0 + t\phi_1], [(1-s)\phi_0 + s\phi_1]) \\ &\quad= \inf_h \; ||((1-t)\phi_0 + t\phi_1) - ((1-s)\phi_0 + s\phi_1) \circ h||_\infty, \end{aligned}$$

where h ranges over all reparameterizations of $[0,1]$. Continuing, we find:

$$\begin{aligned} &d_{FG}([\Gamma_t], [\Gamma_s]) \\ &\quad= \inf_h \; ||(s-t)\phi_0 + (t-s)\phi_1 \circ h||_\infty \\ &\quad= |t-s| \inf_h \; || - \phi_0 + \phi_1 \circ h||_\infty \\ &\quad< \delta \cdot \inf_h \; || - \phi_0 + \phi_1 \circ h||_\infty \\ &\quad= \varepsilon, \end{aligned}$$

by definition of δ.

And so, we have shown that Γ satisfies the the ε-δ definition of continuity. In extended metric spaces, the ε-δ definition of continuity is equivalent to topological continuity (e.g., see proof in [23, Lemma 7.5.7] for metric spaces). Thus, we conclude that linear interpolation between graphs in $\mathcal{G}_\mathcal{C}G$ is continuous. □

Setting $G = [0,1]$, an identical argument shows that linear interpolation between paths in $\Pi_\mathcal{C}$ is also continuous.

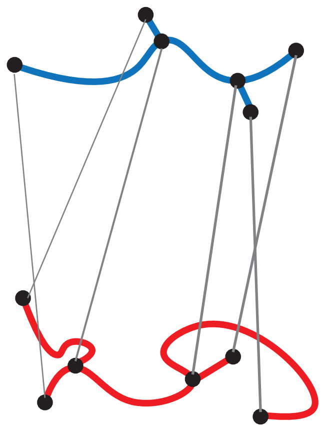


Figure 4: The interpolation between two embeddings of a graph in $\mathbb{R}^n$. For simplicity, we show the interpolation between the vertices in the embeddings, and the interpolation between edges is inferred accordingly.

Corollary 36 (Linear Interpolation between Paths) *For all $[\gamma_0], [\gamma_1] \in \Pi_\mathcal{C}$, the linear interpolation from γ_0 to γ_1 is continuous.*

B.2 Paths Between Immersions in Greater Detail

This subsection includes additional details to maintain local injectivity for an arbitrary path $\Gamma : [0,1] \to \Pi_\mathcal{I}$. We begin by examining the case where pausing occurs on the closed interval $[a,b] \subset [0,1]$ in the domain of an immersed path $\gamma_t \in \Gamma_t$, and the interval includes either 0 or 1. We subvert this by finding an alternate (but 'close') path Γ^*.

Lemma 37 (Pausing at Endpoints) *Let $[\gamma_0], [\gamma_1] \in \Pi_\mathcal{I}$, and let $\Gamma : [0,1] \to \Pi_\mathcal{C}$ be a path in $\Pi_\mathcal{C}$ starting at γ_0 and ending at γ_1. Suppose that there exists an interval $[t_1, t_2] \subset [0,1]$ such that for $t \in [0,1] \setminus (t_1, t_2)$, Γ_t is an immersion and, for $t \in (0,1)$, Γ_t has a single pause (and no other violations of local injectivity). Then, there exists an alternate path Γ^* in $\Pi_\mathcal{C}$ starting at γ_0 and ending at γ_1 such that Γ^* is a path in $\Pi_\mathcal{I}$.*

Proof. We use the same idea as in Lemma 15, but instead stretch the unit interval into only one side of the original domain of the path Γ_t. That is:

$$\Gamma^*_t(x) := \begin{cases} \Gamma_t(x \cdot a) & \text{if } b = 1 \\ \Gamma_t((x-b)\cdot(1-b)+b) & \text{if } a = 0 \end{cases} \tag{3}$$

If Γ_t pauses on $[a,1]$, we know that the we can focus on the image of $[0,a]$. Likewise, if Γ_t pauses on $[0,b]$, we turn to the image of $[b,1]$. Then, replace Γ_t that pauses with the newly defined Γ^*_t. And so, we define a new map $\Gamma^* \colon [0,1] \to \Pi_{\mathcal{C}}$ as follows:

$$\Gamma^*(t) := \begin{cases} \Gamma_t & \text{if } t \notin (t-\varepsilon, t+\delta) \\ \Gamma^*_t & \text{if } t \in (t-\varepsilon, t+\delta) \end{cases} \tag{4}$$

Indeed, it is easy to verify that each Γ^*_t preserves local injectivity so $\Gamma^*_t \in \Pi_{\mathcal{I}}$. Moreover, Γ^* is continuous. □

We now examine the case when linear interpolation results in a singleton, which causes a degeneracy in spaces of immersions. We give a maneuver to subvert this for paths.

Lemma 38 (Dodging Singletons) *Let $[\gamma_0], [\gamma_1] \in \Pi_{\mathcal{I}}$, and let $\Gamma : [0,1] \to \Pi_{\mathcal{C}}$ be a linear interpolation from γ_0 to γ_1. Let $t \in [0,1]$ such that $\Gamma(t)$ is a constant map, forcing $\Gamma(t) \notin \Pi_{\mathcal{I}}$. We can avoid this total degeneracy by rotating $\Gamma(t)$.*

Proof. Linear interpolation of γ_0 to γ_1 produces a singleton if the two equivalence classes of paths are colinear with reversed orientation. Hence, if Γ_t degenerates to a constant map, there exists sufficiently small $\epsilon > 0$ to continuously rotate $\Gamma(t-\epsilon)$ by π without forcing $d_{FP}(\Gamma(t), \gamma_1) > d_{FP}(\gamma_0, \gamma_1)$. Thereby reversing the orientation of $\Gamma(t+\epsilon)$, and avoiding the constant map for any $\gamma_t \in \Gamma_t$. See Figure 5 for an example. □

We now consider the case of backtracking during linear interpolation, which violates local injectivity. We introduce a maneuver to solve this potential degeneracy in spaces of immersions.

Lemma 39 (The Q-Tip Maneuver) *Let $[\gamma_0], [\gamma_1] \in \Pi_{\mathcal{I}}$, and let $\Gamma : [0,1] \to \Pi_{\mathcal{C}}$ be a linear interpolation from γ_0 to γ_1. Let $t \in [0,1]$ such that $\Gamma(t)$ creates backtracking for some $\Gamma(t)$. Inflating a ball about the critical backtracking point corrects this violation of injectivity.*

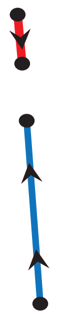
Proof. In the scenario of a backtracking event, local injectivity is only violated at the exact critical point $\Gamma_t(x)$ for $x \in [0,1]$ where backtracking occurs. For sufficiently small $\varepsilon, \delta > 0$, continuously inflate a ball of radius δ about $\Gamma_{t-\varepsilon}(x)$ such that $d_{FP}([\Gamma t - \epsilon], [\gamma_1])$ remains fixed, creating the path Γ^*_t with a ball replacing the critical point, so that $\Gamma^*_t \in \widetilde{\Pi_{\mathcal{I}}}$. Then, replace any backtracking Γ_t with the corresponding Γ^*_t. For every $t \in [0,1]$ it holds that $\Gamma_t \in \Pi_{\mathcal{I}}$, and by the continuity of the inflation, Γ remains continuous. For an example of this maneuver, see Figure 6b. □

B.3 Balls of Path Embeddings in Greater Detail

In what follows, we elaborate on counterexamples for the path-connectivity of balls in $\Pi_{\mathcal{E}}$ and $\mathcal{G}_{\mathcal{E}}$. We begin with a counterexample for path embeddings in $\mathbb{R}^2$.

We continue with a brief description of counterexamples for the path-connectivity of embedded paths in $\mathbb{R}^3$.

(a) Paths with reversed orientation


(b) Interpolate

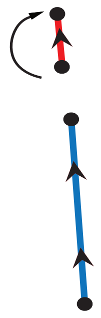
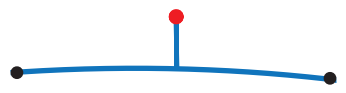
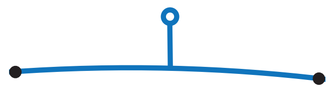

(c) Rotate when sufficiently close

Figure 5: For colinear paths with opposing orientation, rotating by π avoids degenerating to the constant map, keeping Γ in $\Pi_{\mathcal{I}}$. Moreover, rotation with sufficiently small Fréchet distance maintains the path-connectivity of balls.



(a) Example path with backtracking



(b) Inflate the critical backtracking point

Figure 6: Reconcile forced backtracking along a path by inflating a ball about the critical backtracking point, thereby maintaining local injectivity.

Lemma 40 (3d Balls in $\Pi_\mathcal{E}$) *If $n = 3$, metric balls in the space $(\Pi_\mathcal{E}, d_{FP})$ of embedded paths in $\mathbb{R}^n$ are not path-connected.*

Proof. Metric balls are not in general path-connected in three dimensions. For a simple counterexample, suppose γ_0 comprises a loop in $\mathbb{R}^3$, where a segment crossed on top of itself, avoiding self-intersection by some small distance δ, with long tails at either end of the crossing of length 2δ. Suppose also that γ_1 comprises the mirror image of γ_0. Then, $d_{FP} = \delta$, but it is not possible to construct a path from γ_0 to γ_1 without increasing the Fréchet distance between the two, since γ_0 must conduct a self-crossing, which increases the Fréchet distance by at least 2δ. Again, see Figure 3 □

We conclude with additional details demonstrating the path-connectivity of balls for embedded paths in $\mathbb{R}^4$ or higher.

Lemma 41 (Balls in $(\Pi_\mathcal{E}, d_{FP})$, $n \geq 4$) *If $n \geq 4$, balls in the metric space of embedded paths $(\Pi_\mathcal{E}, d_{FP})$ in $\mathbb{R}^n$ are path-connected.*

Proof. Let $[\gamma_0], [\gamma_1], [\gamma_2] \in \Pi_\mathcal{E}$ in the ambient space $\mathbb{R}^n$, for $n \geq 4$. Let $\delta > 0$, and $\mathbb{B} := \mathbb{B}_{d_{FG}}([\gamma_0], \delta) \subset \Pi_\mathcal{E}$. Since all topological knots are represented equivalently in only three dimensions, without loss of generality, we consider the projections of every $\gamma_0 \in [\gamma_0]$, $\gamma_1 \in [\gamma_1]$, and $\gamma_2 \in [\gamma_2]$ in $\mathbb{R}^3$. Construct a continuous $\Gamma : [0,1] \to \Pi_\mathcal{E}$ by the linear interpolation from $\Gamma(0) = \gamma_1$ to $\Gamma(1) = \gamma_2$. By the rectifiability of the embeddings γ_1 and γ_2, the interpolation must reduce $d_{FP}(\gamma_1, \gamma_2)$ by some $\epsilon > 0$ before a self-crossing is required in the image of Γ_t at some $t \in [0,1]$.

At t, conduct a self-crossing by perturbing Γ_t in the fourth dimension by no more than $\epsilon/2$. This increases $d_{FP}(\Gamma_t, \gamma_2)$ by no more than $\epsilon/2$. Hence, $d_{FP}(\Gamma_t, \gamma_2)$ is either strictly decreasing as $t \to 1$, or necessarily satisfies $d_{FP}(\Gamma_t, \gamma_2) \leq \delta - \epsilon/2$ for $\epsilon > 0$. This is to say, for all $t \in [0,1]$, $d_{FP}(\Gamma_t, \gamma_2) \leq \delta$, and $\Gamma_t \in \mathbb{B}$. Hence, metric balls in the space are path-connected. □

Optimal Polyline Simplification under the Local Fréchet Distance in (Near-)Quadratic Time

Peter Schäfer* Sabine Storandt* Johannes Zink†

Abstract

Given a polyline on n vertices, the polyline simplification problem asks for a minimum-size subsequence of the vertices defining a new polyline whose distance to the original polyline is bounded by a given threshold. As distance measure, we employ the frequently used local Fréchet distance. The well-known Imai–Iri algorithm solves this problem to optimality in $\mathcal{O}(n^3)$ time using linear space. Recently, Buchin et al. [ESA'22] presented the first subcubic algorithm with a running time and space consumption in $\mathcal{O}(n^{5/2+\varepsilon})$ for any $\varepsilon > 0$. We show that there is an algorithm with a running time in $\mathcal{O}(n^2 \log n)$ using only linear space. Moreover, we conduct an extensive experimental evaluation on real-world trajectories. We observe that our algorithm requires only a running time in $\mathcal{O}(n^2)$ on all of these instances.

1 Introduction

Polyline simplification owes its relevance to various applications, such as processing of vector graphics [29, 32], robotics [25, 13], trajectory clustering [7], shape analysis [24], data compression [23], curve fitting [27], and map visualization [2, 3, 16, 20, 30]. The task is to replace an n-vertex polyline with a minimum-size subsequence of its vertices while keeping the input and the output polyline sufficiently similar. The similarity is governed by a distance threshold δ. To determine the similarity, the Hausdorff and the Fréchet distance are the most commonly used measures. Throughout the paper, distance measures are applied *locally*, i.e., the distance between each line segment of the output and the part of the input polyline it bridges (instead of the polylines as a whole) must not exceed δ. Local measures allow for a clearer mapping between input and output polyline and they reflect the course of the original polyline better.

Related Work. For the local Hausdorff distance, the Imai–Iri algorithm [19] guarantees a running time in $\mathcal{O}(n^3)$. Melkman and O'Rourke [22] improved this to a running time in $\mathcal{O}(n^2 \log n)$, which was further reduced to $\mathcal{O}(n^2)$ by Chan and Chin [11].

A drawback of using the Hausdorff distance is that it does not reflect the similarity of the courses of two polylines. In contrast, the Fréchet distance measures the maximum distance between two polylines while traversing them in parallel and is therefore often regarded as the better suited measure for polyline similarity. For the local Fréchet distance, though, the cubic running time of the Imai–Iri algorithm (shown by Godau [15] in 1991) was a longstanding bound and has still been quoted as the state of the art in recent publications [8, 29]. Agarwal et al. [1] posed the problem of whether there exists a subcubic algorithm for polyline simplification under the local Fréchet distance as an open question, which was answered positively recently by Buchin et al. [10]. They describe a data structure that outputs the Fréchet distance between any line segment and any subpolyline of a preprocessed polyline in $\mathcal{O}(\sqrt{n} \log^2 n)$ time. Using this data structure to test, for each pair of vertices, whether they can be connected by a segment in the output polyline, a polyline simplified optimally can be computed in $\mathcal{O}(n^{5/2+\varepsilon})$ time (and space) for any $\varepsilon > 0$. We remark that this data structure is quite sophisticated and more powerful than required for polyline simplification.

The most practically relevant setting for polyline simplification is to consider two-dimensional curves in the Euclidean plane (i.e., under the L_2 norm). However, the problem was also studied in higher dimensions $d > 2$ and under different norms. The Imai–Iri algorithm works in $\mathbb{R}^{d \geq 2}$ with the running time only increasing polynomially in d. Bringmann and Chaudhury [8] have proven conditional lower bounds for simplification in $\mathbb{R}^d$ under the local Hausdorff/Fréchet distance. For L_p with $p \in [1, \infty), p \neq 2$, algorithms with a running time subcubic in n and polynomial in d are ruled out unless the $\forall\forall\exists$-OV hypothesis fails. However, for small values of d (which are of high practical relevance), faster algorithms are possible, as evidenced by a $\mathcal{O}(d 2^d n^2)$ time algorithm for the local Hausdorff distance under the L_1 norm [5]. For L_2 and L_∞, the best currently known conditional lower bound was proven by Buchin et al. [9]. It rules out algorithms with a subquadratic running time in n and polynomial running time in d unless SETH fails.

For a faster runtime, alternative constraints on the output polyline, heuristics, and approximation algorithms have been investigated. For instance, Durocher et al. [14] require the maximum number of intersections

*Universität Konstanz, Germany
†Universität Würzburg, Germany

between input and output polyline or Visvalingam and Whyatt [31] refer to the triangular area a vertex adds. The Ramer–Douglas–Peucker algorithm [27, 12], one of the most simple and widely used heuristics, computes a simplified polyline under the local Hausdorff distance in $\mathcal{O}(n \log n)$ time [18] and under the local Fréchet distance in $\mathcal{O}(n^2)$ time [29], but without any guarantee regarding the solution size. Agarwal et al. [1] presented an approximation algorithm with a running time of $\mathcal{O}(n \log n)$ that works for any L_p norm and generalizes to $\mathbb{R}^d$. Under the local Fréchet distance, the simplification size does not exceed the optimal simplification size for $\delta/2$.

The problem variant where the requirement is dropped that all vertices of the simplification must be vertices of the input polyline is called a *weak* simplification. Guibas et al. [17] showed that an optimal weak simplification under the (non-local) Fréchet distance can be computed in $\mathcal{O}(n^2 \log^2 n)$ time. Later Agarwal et al. [1] gave an $\mathcal{O}(n \log n)$-time approximation algorithm violating the distance threshold δ by a factor of at most 8 (see also Van de Kerkhof et al. [28]).

Our Contribution. We present an algorithm for polyline simplification under the local Fréchet distance running in $\mathcal{O}(n^2 \log n)$ time and linear space. Our algorithm builds upon the Melkman–O'Rourke algorithm [22], which exploits the geometric properties of the local Hausdorff distance using cone-shaped *wedges* and a *wavefront* of circular arcs to determine all possible segments of the output polyline. We adapt both of these concepts to the local Fréchet distance. We study the properties of the resulting wavefront and explain how to maintain our wavefront data structure efficiently.

As our main result, we prove that the asymptotic running time does not increase with our modifications. This is a large improvement compared to the cubic running time of the Imai–Iri algorithm and also to the currently stated best runtime bound of $\mathcal{O}(n^{5/2+\varepsilon})$ [10]. It is also faster than the $\mathcal{O}(n^2 \log^2 n)$-time weak-simplification algorithm by Guibas et al. [17, Theorem 14]. However, we remark that parts of their algorithm (Theorem 7, Lemma 8, Lemma 9) can be used to obtain similar results. Yet, their procedure is more complicated since it maintains geometric information only needed for weak simplifications. Our algorithm is tailored to polyline simplification under the local Fréchet distance, and thus cleaner, as well as easier to implement and analyze.

As one result of our in-depth analyis, we show that under the L_1 and L_∞ norm, the wavefront has constant complexity, improving the running time to $\mathcal{O}(n^2)$.

Finally, we investigate real-world trajectories. Across all instances, the wavefront always has very small size, resulting in an $\mathcal{O}(n^2)$ runtime. This matches the asymptotic running time for the local Hausdorff distance and of the wide-spread Ramer–Douglas–Peucker heuristic.

2 Preliminaries

A *polyline* is a series of line segments defined by a sequence of points $L = \langle p_1, \ldots, p_n \rangle$ in the plane called *vertices*. By n, we denote the *length* of a polyline. For $1 \leq i \leq j \leq n$, we let $L[p_i, p_j] := \langle p_i, \ldots, p_j \rangle$, i.e., the subpolyline of L starting p_i, ending at p_j, and including all vertices in between in order. The continuous (but not smooth) curve induced by the vertices of L is denoted as $c_L \colon [1, n] \to \mathbb{R}^2$ with $c_L \colon x \mapsto (\lfloor x \rfloor + 1 - x) p_{\lfloor x \rfloor} + (x - \lfloor x \rfloor) p_{\lceil x \rceil}$.

Polyline Simplification

Input: A polyline L, a distance measure d_X comparing two polylines, and a distance threshold δ.
Output: Return a minimum-size subsequence S (*simplification*) of L (*original* polyline) such that $p_1, p_n \in S$ and $d_X(L, S) \leq \delta$.

As distance measure d_X, commonly the local Hausdorff or the local Fréchet distance is employed.

Fréchet Distance of Two Polylines

Input: Polylines $L = \langle p_1, \ldots, p_n \rangle$, $L' = \langle q_1, \ldots, q_m \rangle$.
Output: $d_\mathrm{F}(L, L') := \inf_{\alpha, \beta} \max_{t \in [0,1]} d(c_L(\alpha(t)), c_{L'}(\beta(t)))$, where $\alpha \colon [0,1] \to [1,n]$ and $\beta \colon [0,1] \to [1,m]$ are continuous non-decreasing functions s.t. $\alpha(0) = \beta(0) = 1$, $\alpha(1) = n$, and $\beta(1) = m$.

We measure only the *local* Fréchet distance, i.e., the maximum of the Fréchet distance between a line segment $\langle p_i, p_j \rangle$ of the simplification and its corresponding subpolyline $L[p_i, p_j]$ in the original polyline. When using a local distance measure, we can tell for each pair of vertices $\langle p_i, p_j \rangle$ independently whether a simplification may contain the *shortcut* $\langle p_i, p_j \rangle$ or not. If the distance does not exceed the distance threshold δ, we also call $\langle p_i, p_j \rangle$ a *valid* shortcut. Note that $\langle p_i, p_{i+1} \rangle$ is always a valid shortcut for any $i \in \{1, \ldots, n-1\}$.

In the definition of the Fréchet distance, we can choose how the distance between two points is determined. Typically, a vector norm is used. For $p \in [1, \infty)$, the L_p norm of a two-dimensional vector $x \in \mathbb{R}^2$ is defined as $\|x\|_p := (|x_1|^p + |x_2|^p)^{1/p}$. For $p = 1$, it is called the Manhattan norm, for $p = 2$, the Euclidean norm. For $p \to \infty$, it is called the maximum norm, which is defined as $\max\{x_1, x_2\}$. The *unit circle* in L_p is the set of points within unit distance to the origin. While this unit is conventionally set to 1, we use δ here instead as this allows for easier argumentation when using distance bound δ. For L_1 and L_∞, the unit circle actually forms a square with side length $\sqrt{2}\delta$ and 2δ, respectively. For L_2, it is a (geometric) circle with radius δ. For p between 2 and ∞, it forms a supercircle which for larger p approaches a square. We refer to a contiguous subset of the boundary of a unit circle in L_p as an *arc*.

Imai–Iri Algorithm. The polyline simplification algorithm by Imai and Iri [19] proceeds in two phases. In the first phase, the *shortcut graph* is constructed. This graph has a node for each vertex of L and it has an edge between two nodes iff there is a valid shortcut between the two corresponding vertices of L. For the Hausdorff/Fréchet distance, we can check in $\mathcal{O}(n)$ time if the distance between a line segment and an $\mathcal{O}(n)$-vertex polyline exceeds δ [4]. Thus, the total runtime of the first phase amounts to $\mathcal{O}(n^3)$. In the second phase, a shortest path from p_1 to p_n is computed in the shortcut graph, which can be accomplished in $\mathcal{O}(n^2)$ time.

In a naive implementation, the space consumption is in $\mathcal{O}(n^2)$. However, it is not necessary to first construct the full shortcut graph and to compute the shortest path subsequently. Instead, the space consumption can be reduced to $\mathcal{O}(n)$ by interleaving the two phases as follows: For p_i, the shortest path distance d_i from p_i to p_n via shortcuts can be computed in linear time by considering all valid shortcuts $\langle p_i, p_j \rangle$ to vertices p_j with $j > i$ and setting $d_i = 1 + \min_{\langle p_i, p_j \rangle} d_j$. Hence, if we traverse the vertices in reverse order, all we need to keep in memory are the distance values for already processed vertices and the shortcuts of the currently considered vertex.

Melkman–O'Rourke Algorithm. Since in the Imai–Iri algorithm the construction of the shortcut graph dominates the runtime, accelerating this phase leads to an overall improvement. Melkman and O'Rourke [22] introduced a faster technique to compute the shortcut graph for the local Hausdorff distance. Starting once at each vertex p_i for $i \in \{1, \ldots, n\}$, they traverse the rest of the polyline vertex by vertex in $\mathcal{O}(n \log n)$ time to determine all valid shortcuts.

To this end, they maintain a cone-shaped region called *wedge* in which all valid shortcuts lie. When traversing the polyline, the wedge may become narrower iteratively. Moreover, they maintain a *wavefront*,[1] which is a sequence of circular arcs of unit circles. The wavefront subdivides the wedge into two regions – a valid shortcut $\langle p_i, p_j \rangle$ has the endpoint p_j in the region not containing p_i. Thus, a valid shortcut needs to cross the wavefront. The wavefront has size in $\mathcal{O}(n)$ and is stored in a balanced search tree (they use an augmented 2-3-tree) such that querying and updating operations can be performed in amortized $\mathcal{O}(\log n)$ time.

Containment in the wedge can be checked in $\mathcal{O}(1)$ time and the position of a vertex relative to the wavefront can be determined in $\mathcal{O}(\log n)$ time. The wedge can be updated in $\mathcal{O}(1)$ time. Updating the wavefront may involve adding an arc and removing several arcs. It is a crucial observation [22] that the order of arcs on the wavefront is reverse to the order of the corresponding unit circle centers with respect to the angle around p_i. This allows for binary search in $\mathcal{O}(\log n)$ time to locate a new arc within the wavefront. Although a linear number of arcs may be removed from the wavefront in a single step, over all steps any arc is removed at most once. Amortized, this yields a running time of $\mathcal{O}(n \log n)$ per starting vertex p_i and $\mathcal{O}(n^2 \log n)$ in total.

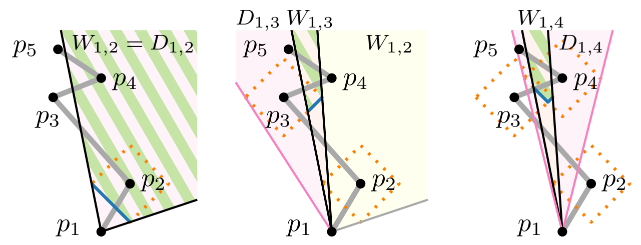
Algorithm by Guibas et al. Guibas et al. [17] study weak polyline simplification. There, one is given an n-vertex polyline $L = \langle p_1, \ldots, p_n \rangle$ and a distance threshold δ, and the objective is to compute any polyline $S = \langle q_1, \ldots, q_m \rangle$ of smallest possible length m that hits all unit circles around the vertices in L in the given order, which they call *ordered stabbing*. To additionally have Fréchet distance at most δ between L and S, each vertex q_j of S needs to be in distance $\leq \delta$ to some point of c_L. They describe an $\mathcal{O}(n^2 \log n)$-time 2-approximation algorithm and a dynamic program solving this problem exactly in $\mathcal{O}(n^2 \log^2 n)$ time.

Both algorithms essentially rely on a subroutine to decide whether there exists a line ℓ (a *stabbing line*) that intersects a given set of n ordered unit circles $\langle C_1, \ldots, C_n \rangle$ such that ℓ hits some points $\langle r_1, \ldots, r_n \rangle$ with $r_i \in C_i$ for $i \in \{1, \ldots, n\}$ in order [17, Def. 4]. This subroutine runs in $\mathcal{O}(n \log n)$ time [17, Lemma 9]. It is based on an algorithm computing iteratively two hulls and two limiting lines through the unit circles that describe all stabbing lines [17, Algorithm 1]. They also maintain the wavefront as described in the Melkman–O'Rourke algorithm. However, they add an update step to ensure that the stabbing line respects the order of the unit circles. We use conceptually the same update step and explain it in more detail in Secs. 3 and 4.

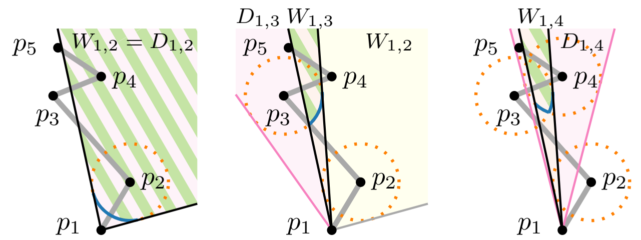
Definitions and Notation. The following definitions are illustrated in Fig. 1. When starting at p_i and encountering p_j during the traversal, we denote by $D_{i,j}$ the *local wedge* of p_i and p_j that is the area between the two tangential rays of the unit circle around p_j emanating at p_i. The (global) wedge $W_{i,j}$ is an angular region having its origin at p_i. We define $W_{i,i}$ to be the whole plane and each $W_{i,j}$ for $j > i$ is essentially the intersection of all local wedges up to $D_{i,j}$. We remark that, we apply an extra update step described in Sec. 3 specific to the Fréchet distance, which may narrow the wedge when obtaining $W_{i,j}$ from $W_{i,j-1}$. Therefore, $W_{i,j} \subseteq \bigcap_{k \in \{i+1, i+2, \ldots, j\}} D_{i,k}$ holds. We give a precise inductive definition of the wedge $W_{i,j}$ when we describe the algorithm in Sec. 3.

Let C_j be the unit circle around p_j and let l_j (r_j) be the left (right) tangential point of C_j and $D_{i,j}$. Between l_j and r_j, there are two arcs of C_j – the *bottom arc* and

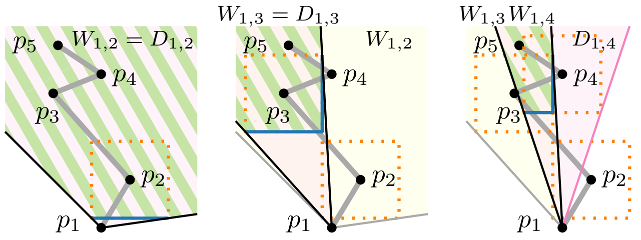
[1]Melkman and O'Rourke [22] use the term *frontier* instead of wavefront. Within the cone, they only call the region on the other side of the frontier *wedge* and they call the associated data structure *wedge data structure*. Our notation to call the whole cone *wedge* is in line with the algorithm by Chan and Chin [11].



(a) L_1 norm: the unit circles are squares of side length $\sqrt{2}\delta$.



(b) L_2 norm: the unit circles are circles of radius δ.



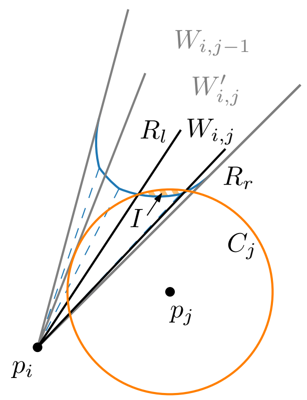
(c) L_∞ norm: the unit circles are squares of side length 2δ.

Figure 1: Iterative construction of the wedge: the local wedges $D_{1,2}$, $D_{1,3}$, and $D_{1,4}$ (pink) bound the wedges $W_{1,2}$, $W_{1,3}$, and $W_{1,4}$ (yellow). The wavefront consists of unit circle arcs (blue). Above the wavefront is the valid region (hatched green), where a subsequent vertex p_j of a valid shortcut $\langle p_1, p_j \rangle$ lies. Here, $\langle p_1, p_5 \rangle$ is a valid shortcut in L_∞ while in L_1 and L_2 it is not.

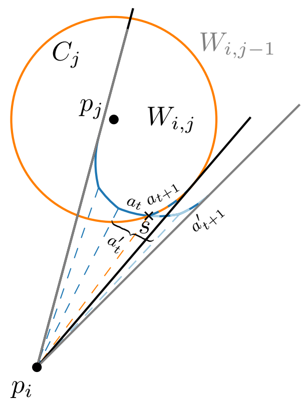
the *top arc*[2]. Any ray emanating at p_i intersects the bottom and top arc at most once each. We call the bottom arc of C_j between l_j and r_j the *wave* of $D_{i,j}$. We call the region within $D_{i,j}$ and above and on its wave the *local valid region* of $D_{i,j}$, and the region within $W_{i,j}$ and above and on the wavefront the *valid region* of $W_{i,j}$ (for $W_{i,i}$ the whole plane). The wavefront is defined inductively: the *wavefront* of $W_{i,j}$ is the boundary of the intersection of the valid region of $W_{i,j-1}$ and the local valid region of $D_{i,j}$ within $W_{i,j}$ excluding the boundary of $W_{i,j}$. Intuitively, it is the wavefront of $W_{i,j-1}$ within $W_{i,j}$ where we cut out the bottom arc of C_j.

We provide proofs for claims with ($\star$) in the appendix.

[2]W.l.o.g., we assume that p_i is below p_{i+1} and therefore at the bottom of a wedge. Moreover, we assume that p_{i+1} has a distance of at least δ to p_i as otherwise, we could ignore all vertices following p_i with distance $\leq \delta$ to p_i since they are in δ-distance to any shortcut $\langle p_i, p_j \rangle$. Note, though, that a vertex p_j with $j > i+1$ could have distance $\leq \delta$ to p_i. Then, we define $D_{i,j}$ as the whole plane and the whole boundary of C_j as its *top arc*.



(a) When we encounter p_j, we update the wedge in two steps even if p_j is outside the wedge.



(b) Vertex p_j contributes an arc to the wavefront. Here, $\langle p_i, p_j \rangle$ is a valid shortcut.

Figure 2: Updating the wedge and its wavefront in L_2.

3 A Fast Polyline Simplification Algorithm.

Next, we describe our algorithm for polyline simplification under the local Fréchet distance running in near-quadratic time by means of Melkman and O'Rourke [22] and integrating ideas from Guibas et al. [17].

Outline. As all Imai–Iri based algorithms, we build the shortcut graph by traversing the given polyline n times – starting once from each vertex p_i for $i \in \{1, \ldots, n\}$ and determining all shortcuts starting at p_i. For each p_i, we construct a wedge with a wavefront, whose properties are analyzed in more detail in Sec. 4.

Next, we describe how to determine, for each vertex p_i, the set of subsequent vertices to which p_i has a valid shortcut. We traverse the polyline in order $p_{i+1}, p_{i+2}, \ldots, p_n$. During this traversal, we maintain the wedge and the wavefront. As in the algorithm by Melkman and O'Rourke, our invariant is that for a valid shortcut $\langle p_i, p_j \rangle$ with $j > i$, p_j has to lie inside the valid region of the wedge $W_{i,j-1}$. In this case, we add the directed edge $p_i p_j$ to the shortcut graph.

Then, regardless of whether $\langle p_i, p_j \rangle$ is a valid shortcut or not, we first update the wedge $W_{i,j-1}$ to an intermediate wedge $W'_{i,j}$ by computing the intersection between $W_{i,j-1}$ and the local wedge $D_{i,j}$, and second, we update $W'_{i,j}$ to the wedge $W_{i,j}$ and we update the wavefront.

This update step is illustrated for the L_2 norm in Fig. 2 and for multiple steps and multiple norms in Fig. 1. For the L_2 norm and for the L_1 and L_∞ norms, we give more detail on this update step in Sec. 4. It works as follows. A valid shortcut $\langle p_i, p_k \rangle$ with $k > j$ needs to go through the intersection region I between the current valid region and the unit circle C_j around p_j. Otherwise, the vertices of the subpolyline from p_i to p_k would be encountered in the wrong order contradicting the definition of the Fréchet distance. Hence, we narrow the intermediate wedge $W'_{i,j}$ such that the rays R_l and R_r emanating at p_i and enclosing I constitute the wedge $W_{i,j}$; see Fig. 2a. Such a narrowing step is also applied by Guibas et al. but not by Melkman and O'Rourke. For the Hausdorff distance, it is irrelevant in which order the intermediate points of a shortcut are encountered by the shortcut segment.

Thereafter, we update the wavefront as Melkman and O'Rourke do. The part of the bottom arc of the unit circle C_j around p_j that is above the current wavefront is included into the wavefront. Pictorially, the wavefront is moving upwards. For an example see Fig. 2b. There, we compute the intersection point s between C_j and the wavefront and replace the arcs a'_t and a'_{t+1} of the wavefront by the arcs a_t (which is a part of a'_t) and a_{t+1} (which is a part of C_j). There can be up to two intersection points between C_j and the wavefront.

If the valid region becomes empty, we abort the search for further shortcuts from p_i.

Correctness. To show that the algorithm works correctly, we prove two things: that all shortcuts the algorithm finds are valid (Lemma 2) and that the algorithm finds all valid shortcuts (Lemma 3). However, we start with an interesting insight specified in Lemma 1.

Lemma 1 ($\star$) *Consider the wavefront of $W_{i,k}$. For every vertex p_j ($i < j \leq k$) whose unit circle C_j contributes an arc of the wavefront of $W_{i,k}$, p_k and the complete wavefront of $W_{i,k}$ lie inside C_j.*

Lemma 2 ($\star$) *Any shortcut found by the algorithm is valid under the local Fréchet distance and any L_p norm with $p \in [1, \infty]$.*

Proof Sketch. Consider a shortcut $\langle p_i, p_k \rangle$ found by the algorithm. We can show that there is a mapping of the vertices $\langle p_{i+1}, \ldots, p_{k-1} \rangle$ onto subsequent points on the line segment $\overline{p_i p_k}$ such that their distance is at most δ. This implies that the Fréchet distance between $\overline{p_i p_k}$ and $L[p_i, p_k]$ is $\leq \delta$. For a vertex $p_{j \in \{i+1,\ldots,k-1\}}$, we choose as point on $\overline{p_i p_k}$, the intersection point of the wavefront of $W_{i,j}$ and $\overline{p_i p_k}$. Due to Lemma 1, the resulting pairs of points have pairwise distance $\leq \delta$ and we can show that they appear monotonously along $\overline{p_i p_k}$. $\square$

Lemma 3 ($\star$) *All valid shortcuts under the local Fréchet distance and any L_p norm with $p \in [1, \infty]$ are found by the algorithm.*

Proof Sketch. For a valid shortcut $\langle p_i, p_k \rangle$ to not be found, p_k needs to lie outside the valid region of $W_{i,k-1}$. Then, by construction, there was a $p_{j \in \{i+1,\ldots,k-1\}}$ that caused us narrowing the wedge or moving the wavefront and by which we can show that the Fréchet distance between $\overline{p_i p_k}$ and $L[p_i, p_k]$ is $> \delta$. $\square$

4 The Wavefront Data Structure.

At the heart of the algorithm lies the maintenance of the wavefront. To show that the algorithm can be implemented to run in $\mathcal{O}(n^2 \log n)$ time, we next analyze the properties of the wavefront and discuss how to store and update it using a suitable (simple) data structure.

Wavefront Maintenance in $\mathbf{L}_2$. The size of the wavefront is in $\mathcal{O}(n)$. This insight relies on the next property.

Lemma 4 ($\star$) *Any unit circle of radius δ intersects the wavefront at most twice.*

From Lemma 4, it follows that in each step, the size of the wavefront increases at most by 2.

Lemma 5 ($\star$) *The wavefront consists of at most $\mathcal{O}(n)$ arcs under any $L_{p \in (1,\infty)}$ norm.*

As there might be a linear number of arcs on the wavefront, we cannot simply iterate over all arcs in each step of the algorithm since this would require cubic time in total. Similar to Melkman and O'Rourke, we use a balanced search tree (e.g., a red-black tree) where we store the circular arcs of the wavefront. The keys according to which the circular arcs are arranged in the search tree are the angles of their starting points with respect to p_i.

In Appendix B.1, we show how we can update the wavefront in amortized logarithmic time using a simple case distinction. We compute the intersection area I only implicitly. The rough idea is as follows. When we encounter a vertex p_j with unit circle C_j, we keep parts of the wavefront inside C_j and parts of the bottom arc of C_j because, by Lemma 1, the new wavefront needs to lie inside C_j. We conclude our main theorem.

Theorem 6 ($\star$) *An n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_2 (Euclidean) norm in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n)$ space.*

For general L_p norms, we obtain the same, given that we can compute intersection points between a unit circle and a line, and between two unit circles in constant time.

Wavefront Maintenance in $\mathbf{L}_1$ and $\mathbf{L}_\infty$. As the unit circles are square-shaped, the wavefront is initially a subset of the boundary of a rectangle. When we consider the next unit circle, we update the wavefront by intersecting an axis-aligned rectangle and an axis-aligned square, which is again an axis-aligned rectangle.

Lemma 7 ($\star$) *In the L_∞ (L_1) norm, the wavefront consists of either one or two orthogonal line segments, which are horizontal or vertical (rotated by 45 degrees).*

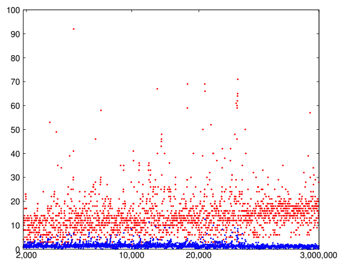
We hence obtain the following theorem.

Theorem 8 *An n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_1 and the L_∞ norm in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space.*

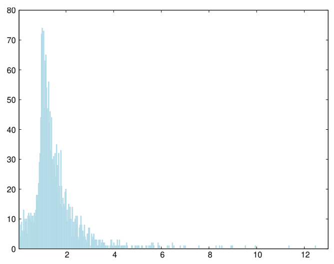
5 Experimental Evaluation

We implemented our algorithm for polyline simplification under the local Fréchet distance in the L_2 norm in C++ [3]. Experiments were conducted on an AMD Ryzen processor at 4.2 GHz without parallelized code. We used sample trajectories gathered from the OSM planet database [26], which holds about 800,000 files containing geographical trajectories given as location sequences. We took three sets of files: (i) the 500 largest files, ranging from 190,000 to 3 million points; (ii) a set of 300 randomly selected files with a bias towards large files; (iii) another set of 100 randomly selected files.

Wavefront Size. For the wavefront to actually grow to linear size, vertices need to be arranged in degenerate patterns, which we do not expect to occur naturally. For the first two trajectory sets, we used as threshold $\delta = 100$ m. For the third set we used several values of δ in the range 1 m–10 km. The average wavefront size was ca. 4, and the maximum rarely exceeded 60 across all sizes and thresholds. For our results, see Figs. 3a and 3b. The wavefront does not seem to grow with the input size at all, confirming our hypothesis that the wavefront size behaves benign on real-world inputs empirically.



(a) Avg. (blue) and max. (red) wavefront size (y-axis) relative to the number of vertices (x-axis).



(b) Frequency (y-axis) of the average wavefront size (x-axis).

Figure 3: Size of the wavefront in real-world instances.

Having consistently small wavefronts implies that the practical running times are closer to $\mathcal{O}(n^2)$ than to $\mathcal{O}(n^2 \log n)$. Also, it allows us to use linear search, which becomes competitive to, and often even faster than the binary search approach in our case distinction. We conducted more detailed experiments on both search strategies and appropriate data structures; see Appendix C.1.

Comparative Performance. We compare the performance of our algorithm to the Imai–Iri algorithm [19] and the approximation algorithm by Agarwal et al. [1].

The cubic running time of the Imai–Iri algorithm does not only occur on certain instances but on all instances, which makes it infeasible to compute simplifications for large input polylines. Hence, we restricted the number of vertices to 11,000; see Fig. 4. The Imai–Iri algorithm needed about 80 ms–252 s, while our algorithm was some orders of magnitude faster (2.5 ms–1.5 s). The results confirm that our algorithm is vastly superior already for small instances. On very large instances with $n \geq 800,000$, our algorithm took up to 13 minutes (4.3 min with multi-core support), while Imai–Iri is infeasible.

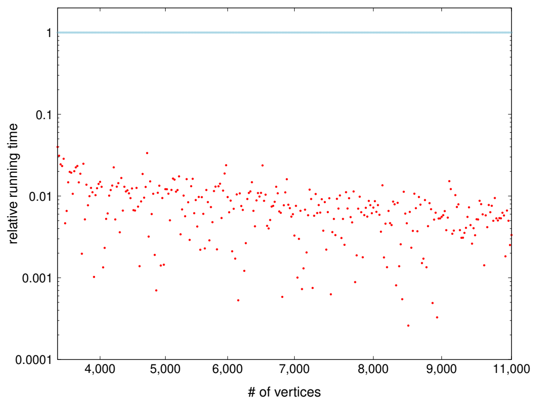


Figure 4: Running times of our algorithm (red dots) relative to the running times of the Imai–Iri algorithm (blue baseline at 1). Logarithmic scale on the y-axis.

Finally, we compare with the approximation algorithm of Agarwal et al. [1]. They replace the shortest-path search of the Imai–Iri algorithm by a greedy approach: at a "greedy" step it searches for the next valid shortcut $\langle p_i, p_k \rangle$ such that $\langle p_i, p_{k+1} \rangle$ is no valid shortcut. The running time of this algorithm is in $\mathcal{O}(n \log n)$. The simplification is guaranteed to be not larger than the optimal size for $\delta/2$. As expected, based on the theoretical runtime bounds, the approximation algorithm is considerably faster than our algorithm but the simplifications are about 20–40 % worse in size; see Fig. 5.

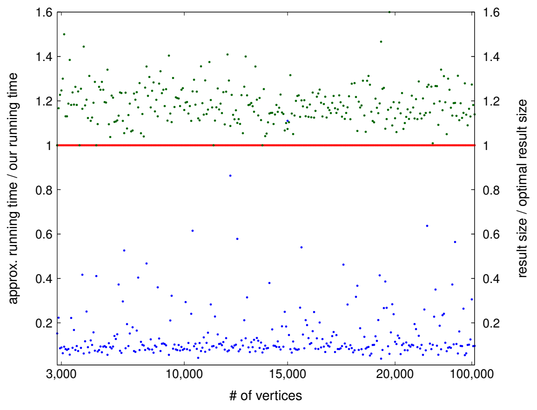


Figure 5: Running time (blue) and result size (green) of the approximation algorithm by Agarwal et al. [1], relative to our algorithm (red baseline at 1).

[3]Source code is available at
`https://gitlab.inf.uni-konstanz.de/ag-storandt/subwaves-public`

References

[1] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005. doi:10.1007/s00453-005-1165-y.

[2] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. Fréchet distance-based map construction algorithm. In *Map Construction Algorithms*, pages 33–46. Springer, 2015. doi:10.1007/978-3-319-25166-0_3.

[3] M. Ahmed and C. Wenk. Constructing street networks from GPS trajectories. In *Proc. 20th Annual European Symposium on Algorithms (ESA'12)*, pages 60–71. Springer, 2012. doi:10.1007/978-3-642-33090-2_7.

[4] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

[5] G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica*, 33(2):150–167, 2002. doi:10.1007/s00453-001-0096-5.

[6] L. Barth. Ygg binary search tree library, 2020. URL: https://github.com/tinloaf/ygg.

[7] M. Brankovic, K. Buchin, K. Klaren, A. Nusser, A. Popov, and S. Wong. (k, l)-medians clustering of trajectories using continuous dynamic time warping. In *Proc. 28th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 99–110, 2020. doi:10.1145/3397536.3422245.

[8] K. Bringmann and B. R. Chaudhury. Polyline simplification has cubic complexity. *Journal of Computational Geometry*, 11(2):94–130, 2021. doi:10.20382/jocg.v11i2a5.

[9] K. Buchin, M. Buchin, M. Konzack, W. Mulzer, and A. Schulz. Fine-grained analysis of problems on curves. In *Proc. 32nd European Workshop on Computational Geometry (EuroCG'16)*, 2016. URL: https://www.eurocg2016.usi.ch/sites/default/files/paper_68.pdf.

[10] M. Buchin, I. van der Hoog, T. Ophelders, L. Schlipf, R. I. Silveira, and F. Staals. Efficient Fréchet distance queries for segments. In *Proc. 30th Annual European Symposium on Algorithms (ESA'22)*, pages 29:1–29:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.29.

[11] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry and Applications*, 6(1):59–77, 1996. doi:10.1142/S0218195996000058.

[12] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973. doi:10.1002/9780470669488.ch2.

[13] R. O. Duda and P. E. Hart. *Pattern classification and scene analysis.* A Wiley-Interscience publication. Wiley, 1973.

[14] S. Durocher, A. Leblanc, J. Morrison, and M. Skala. Robust nonparametric simplication of polygonal chains. *International Journal of Computational Geometry & Applications*, 23(6):427–441, 2014. doi:10.1142/s021819591360011x.

[15] M. Godau. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91)*, pages 127–136, 1991. doi:10.1007/BFb0020793.

[16] M. G. Gruppi, S. V. G. Magalhães, M. V. A. Andrade, W. R. Franklin, and W. Li. An efficient and topologically correct map generalization heuristic. In *Proc. 17th International Conference on Enterprise Information Systems (ICEIS'15)*, pages 516–525, 2015. doi:10.5220/0005398105160525.

[17] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *International Journal of Computational Geometry and Applications*, 3(4):383–415, 1993. doi:10.1142/S0218195993000257.

[18] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. In *Proc. 5th International Symposium on Spatial Data Handling (SDH'92)*, pages 134–143, 1992.

[19] H. Imai and M. Iri. Polygonal approximations of a curve – formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50011-4.

[20] T. Isenberg. Visual abstraction and stylisation of maps. *The Cartographic Journal*, 50(1):8–18, 2013. doi:10.1179/1743277412Y.0000000007.

[21] O. Krzikalla and I. Gaztanaga. Boost.Intrusive, part of Boost C++ Libraries. URL: http://www.boost.org.

[22] A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*, pages 87–95. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50012-6.

[23] N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In *Proc. 9th International Conference on Extending Database Technology (EDBT'04)*, pages 765–782. Springer, 2004. doi:10.1007/978-3-540-24741-8_44.

[24] Y. Min, C. Chen, X. Wang, J. He, and Y. Zhang. SGM: Seed growing map-matching with trajectory fitting. In *Proc. 5th International Conference on Big Data Computing and Communications (BIGCOM'19)*, pages 204–212. IEEE, 2019. doi:10.1109/bigcom.2019.00036.

[25] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis, and R. Siegwart. A comparison of line extraction algorithms using 2d range data for indoor mobile robotics. *Autonomous Robots*, 23(2):97–111, 2007. doi:10.1007/s10514-007-9034-y.

[26] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[27] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972. `doi:10.1016/S0146-664X(72)80017-0`.

[28] M. van de Kerkhof, I. Kostitsyna, M. Löffler, M. Mirzanezhad, and C. Wenk. Global curve simplification. In *Proc. 27th Annual European Symposium on Algorithms (ESA'19)*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ESA.2019.67`.

[29] M. J. van Kreveld, M. Löffler, and L. Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. *Journal of Computational Geometry*, 11(1):1–25, 2020. `doi:10.20382/jocg.v11i1a1`.

[30] M. Visvalingam and J. D. Whyatt. The Douglas-Peucker algorithm for line simplification: Re-evaluation through visualization. *Computer Graphics Forum*, 9(3):213–228, 1990. `doi:10.1111/j.1467-8659.1990.tb00398.x`.

[31] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, 1993. `doi:10.1179/000870493786962263`.

[32] S. Wu and M. R. G. Márquez. A non-self-intersection Douglas-Peucker algorithm. In *Proc. 16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'03)*, pages 60–66. IEEE Computer Society, 2003. `doi:10.1109/sibgra.2003.1240992`.

Appendix

For completeness, we provide proofs and additional but less central content in this appendix.

A Omitted Content from Section 3

We start the appendix with a structural lemma, which we employ for the proofs of Lemmas 1, 4, 10 and 14. It does not yet use the wavefront.

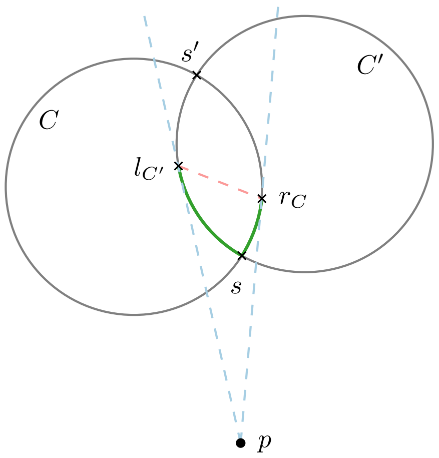
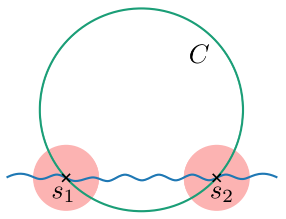


Figure 6: Illustration of the situation described in the proof of Lemma 9.

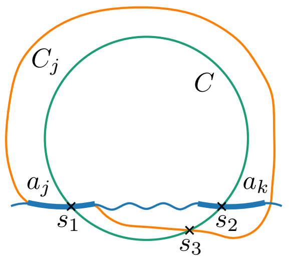
Lemma 9 *Given two unit circles and a point p outside of the unit circles. If the two bottom arcs (with respect to p) intersect, then the second intersection point is between their top arcs.*

Proof. For an illustration of this proof see Fig. 6. Let C and C' be the two unit circles with the center of C being left of the center of C' w.r.t. p. Now the cone between the right tangential from p on C and the left tangential from p on C' contains all of the intersection area of C and C', and hence also both intersection points. We call the tangetial points r_C and $l_{C'}$, respectively. Note that $r_C = l_{C'}$ is excluded as then C and C' would only have a single intersection point. For the intersection point s between the bottom arcs of C and C', we know that the line segment $\overline{ps}$ does not intersect the inner part of any of the two circles by definition of the bottom arc. Hence the ray elongating this line segment has to go through the intersection area of C and C' above s. Therefore, the partial bottom arc of C from s to r_C and the partial bottom arc of C' from s to $l_{C'}$ are both on the boundary of the intersection area. As the intersection area is convex, it means that the line segment $\overline{l_{C'}r_C}$ is fully contained in the intersection area, and the intersection points have to be on opposite sites of the line through $l_{C'}$ and r_C. Accordingly, the second intersection point s' of C and C' then has to lie above $\overline{l_{C'}r_C}$ and is therefore on the respective top arcs of C and C'. $\square$

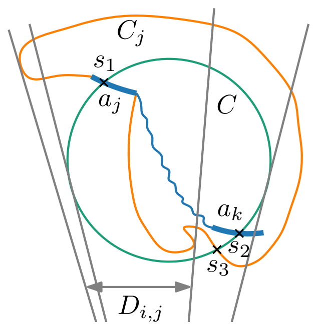
We continue with another structural lemma, which seems rather special at first glance, but we employ it several times,



(a) Intersection pattern of a unit circle C with the wavefront (blue wavy line) that cannot occur.



(b) The unit circle C intersects the wavefront (blue wavy line) twice with its bottom arc.



(c) The unit circle C intersects the wavefront (blue wavy line) with its top and bottom arc.

Figure 7: Sketch for Lemma 10.

e.g., in Appendix B.1, where we analyze the cases for the wavefront maintenance.

Lemma 10 *If a unit circle C intersects the wavefront more than once, then on the left side of the leftmost intersection point s_1 (relative to rays originating in p_i) and on the right side of the rightmost intersection point s_2, C is below the wavefront. In other words, the intersection pattern depicted in Fig. 7a cannot occur.*

Proof. Clearly, if at s_1 the top arc of C intersects the wavefront, then on the left side of s_1, C is below the wavefront. Symmetrically, the same holds for s_2.

Now assume that at s_1 and at s_2, the bottom arc of C intersects the arcs a_j and a_k of the wavefront, respectively. We denote their unit circles by C_j and C_k. W.l.o.g. let C on the left side of s_1 be above the wavefront. By Lemma 1, C_j contains the rest of the wavefront including all of a_k. This means, that C intersects C_j at s_3 in between s_1 and s_2 (potentially $s_2 = s_3$ if $C_j = C_k$); see Fig. 7b. Because the intersection of C at s_2 is with the bottom arc of C, the intersection of C and C_j at s_3 is also with the bottom arc of C. This contradicts Lemma 9.

Finally, assume w.l.o.g. that at s_1 the top arc of C intersects the arc a_j of the wavefront and at s_2 the bottom arc of C intersects the arc a_k of the wavefront; see Fig. 7c. Again by Lemma 1, the unit circle C_j of a_j contains the wavefront including the whole arc a_k. Hence, there is an intersection point s_3 of C and C_j in between s_1 and s_2 (where $s_2 \neq s_3$ and $C_j \neq C_k$ as otherwise C and C_j would have an intersection between their bottom arcs and between a bottom

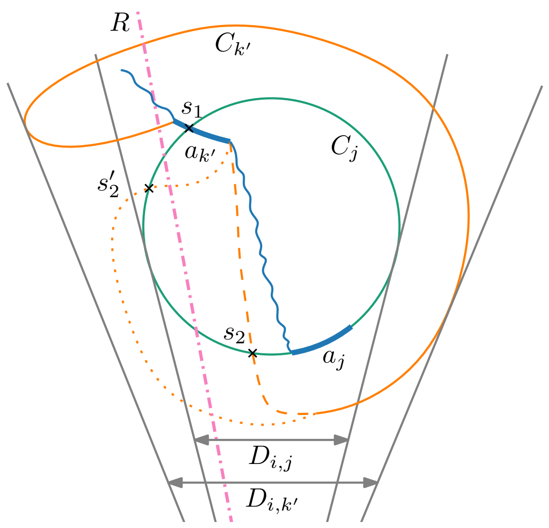


Figure 8: Configuration used to prove Lemma 1. The blue part is the wavefront including the arcs $a_{k'}$ and a_j. The circles C_j and $C_{k'}$ are unit circles and the gray rays indicate the local wedges.

and a top arc). At s_3 there is the bottom arc of C (since later at s_2, there is also the bottom arc of C involved). If C_j also would have its bottom arc at s_3, it would contradict Lemma 9. Therefore, at s_3, there is the top arc of C_j. This however means that s_3 is outside $D_{i,j}$ – a contradiction. $\square$

Lemma 1 ($\star$) *Consider the wavefront of $W_{i,k}$. For every vertex p_j ($i < j \leq k$) whose unit circle C_j contributes an arc of the wavefront of $W_{i,k}$, p_k and the complete wavefront of $W_{i,k}$ lie inside C_j.*

Proof. By construction, C_k contains the complete wavefront of $W_{i,k}$ and, hence, p_k lies also within C_j. It remains to prove that also the wavefront of $W_{i,k}$ lies inside C_j.

We argue that, for all $j \in \{i+1, \ldots, k\}$, the claim is true by considering first all arcs that had been added before and then all arcs that had been added after the arc of C_j had been added to the wavefront.

All arcs $a_{j'}$ on the wavefront belonging to a vertex $p_{j'}$ with $j' < j$ are inside C_j because when the wavefront of $W_{i,j}$ has been constructed, the wavefront of $W_{i,j}$ consisted of arcs of the wave of $D_{i,j}$, i.e., arcs of C_j, and it consisted of arcs of the wavefront of $W_{i,j-1}$ lying inside I, i.e., the intersection between C_j and the valid region of $W_{i,j-1}$[4].

All arcs $a_{k'}$ on the wavefront belonging to a vertex $p_{k'}$ with $j < k' \leq k$ are completely inside C_j because if they were not, there would be an $a_{k'}$ (which is part of the bottom arc of the unit circle $C_{k'}$) that intersects C_j at s_1; see Fig. 8. The intersection at s_1 is with the top arc of C_j as otherwise $a_{k'}$ would be (partially) outside the local valid region of $D_{i,j}$. Still for a_j to be in the local valid region of $D_{i,k'}$, C_j and $C_{k'}$ intersect a second time. We consider two possible cases for a second intersection and denote them by s_2 and s_2'. First, assume that the intersection s_2 is between the bottom arc of $C_{k'}$ and the bottom arc of C_j. This however contradicts Lemma 9 because in s_1, there was already the bottom arc of $C_{k'}$ involved. Hence, the second intersection point is s_2' which is an intersection between the bottom arc of $C_{k'}$ and the top arc of C_j. Then, however, there is a ray R originating in p_i that lies in between s_1 and s_2' and intersects the bottom arc of $C_{k'}$ at least twice – a contradiction. $\square$

[4]We remark that even without the extra narrowing step using I, if C_j contributed an arc of the wavefront of $W_{i,j}$, C_j contained the whole wavefront of $W_{i,j}$.

Lemma 1 directly implies the following lemma.

Lemma 11 *Let q be a point lying on the wavefront of the wedge $W_{i,j}$. Then, $d(p_j, q) \leq \delta$.*

We show one more property before we prove that exactly the valid shortcuts are found by our algorithm.

Lemma 12 *Let R be a ray emanating at p_i and lying inside the wedges $W_{i,j}$ and $W_{i,k}$ for some $i < j < k$. Moreover, let q_j and q_k be the intersection points between R and the wavefronts of $W_{i,j}$ and $W_{i,k}$, respectively. Then, $d(p_i, q_j) \leq d(p_i, q_k)$.*

Proof. Assume for a contradiction that $d(p_i, q_j) > d(p_i, q_k)$. Then, q_k is below the wavefront of $W_{i,j}$ and, hence, q_k does not lie in the valid region of $W_{i,j}$ but in the valid region of $W_{i,k}$. However, the valid region of $W_{i,k}$ is the intersection of the local valid region of $D_{i,k}$ and all previous valid regions including $W_{i,j}$ and, thus, the valid region of $W_{i,k}$ is a subset of $W_{i,j}$. $\square$

Putting Lemma 12 in other words, the wavefront may only move away but never towards p_i.

We are now ready to prove the correctness of our algorithm by the following two lemmas.

Lemma 2 ($\star$) *Any shortcut found by the algorithm is valid under the local Fréchet distance and any L_p norm with $p \in [1, \infty]$.*

Proof. Let $\langle p_i, p_k \rangle$ be a shortcut found by the algorithm. We show that there is a mapping of the vertices $\langle p_{i+1}, p_{i+2}, \ldots, p_{k-1} \rangle$ onto points $\langle m_{i+1}, m_{i+2}, \ldots, m_{k-1} \rangle$, such that $m_j \in \overline{p_i p_k}$ and $d(p_j, m_j) \leq \delta$ for every $j \in \{i+1, \ldots, k-1\}$, and m_j precedes or equals m_{j+1} for every $j \in \{i+1, \ldots, k-2\}$ when traversing $\overline{p_i p_k}$ from p_i to p_k. Clearly, this implies that also the Fréchet distance between each pair of line segments $\overline{p_j p_{j+1}}$ and $\overline{m_j m_{j+1}}$ is at most δ and, hence, $\langle p_i, p_k \rangle$ is a valid shortcut. In the remainder of this proof, we describe how to obtain $m_{i+1}, m_{i+2}, \ldots, m_{k-1} \in \overline{p_i p_k}$. To this end, we consider the wedge $W_{i,j}$ and the corresponding wavefront for each $j \in \{i+1, \ldots, k-1\}$, i.e., for each intermediate step when executing the algorithm. By construction of the algorithm, $\overline{p_i p_k}$ lies inside the wedge $W_{i,j}$ and p_k lies above its wavefront (since p_k lies in the valid region of $W_{i,k-1}$ and, by Lemma 12, the wavefront has never moved towards p_i). Let m_j be the intersection point of $\overline{p_i p_k}$ and the wavefront of $W_{i,j}$. By Lemma 11, $d(p_j, m_j) \leq \delta$. Moreover, by Lemma 12, m_j precedes or equals m_{j+1} for any $j \in \{i+1, \ldots, k-2\}$ when traversing $\overline{p_i p_k}$ from p_i to p_k. $\square$

Lemma 3 (⋆) *All valid shortcuts under the local Fréchet distance and any L_p norm with $p \in [1, \infty]$ are found by the algorithm.*

Proof. Suppose for the sake of a contradiction that there is a valid shortcut $\langle p_i, p_k \rangle$ that was not found by the algorithm.

If p_k lay outside of $\bigcap_{j \in \{i+1, i+2, \dots, k-1\}} D_{i,j}$, then there would be some $p_{j'}$ with $i < j' < k$ such that $d(p_{j'}, \overline{p_i p_k}) > \delta$. So, as in the algorithm by Chan and Chin [11], already the Hausdorff distance requirement would be violated and $\langle p_i, p_k \rangle$ would be no valid shortcut for the local Fréchet distance as well. Hence, p_k lies inside $\bigcap_{j \in \{i+1, i+2, \dots, k-1\}} D_{i,j}$.

Suppose now that p_k lies inside $\bigcap_{j \in \{i+1, i+2, \dots, k-1\}} D_{i,j}$ but outside $W_{i,k-1}$. W.l.o.g. p_k lies to the left of the wedge $W_{i,k-1}$. We know that there is some p_j with $i < j < k$ for which the extra narrowing step from Sec. 3 has been applied such that p_k lies to the left of $W_{i,j}$. For constructing $W_{i,j}$, we have considered the intersection area I between C_j and the wavefront of $W_{i,j-1}$. The left endpoint of I lies on the boundary of $W_{i,j}$ and is the intersection point between C_j and an arc of the wavefront of $W_{i,j-1}$ belonging to a vertex $p_{j'}$ with $i < j' < j$. Now consider the ray R that we obtain by extending $\overline{p_i p_k}$ at p_k. When traversing R, we first enter and leave the interior of C_j before we enter the interior of $C_{j'}$. Hence, the Fréchet distance between $\overline{p_i p_k}$ and $L[p_i, p_k]$ is greater than δ due to the order of $p_{j'}$ and p_j within $L[p_i, p_k]$. Therefore, p_k lies inside $W_{i,k-1}$.

Finally, suppose that p_k lies inside $W_{i,k-1}$ but not in the valid region, i.e., p_k lies below the wavefront of $W_{i,k-1}$. Since p_k is below the wavefront, the line segment $\overline{p_i p_k}$ does not intersect the wavefront (otherwise, we would violate Lemma 13; see below). Again, consider the ray R that we obtain by extending $\overline{p_i p_k}$ at p_k. Let the intersection point of R and the wavefront of $W_{i,k-1}$ be w. The point w lies on an arc of the wavefront. This arc is part of the bottom arc of a unit circle C_j belonging to some p_j with $i < j < k$. Since it is the bottom arc, p_k lies outside C_j and $d(p_k, p_j) > \delta$.

Therefore, p_k lies in the valid region of $W_{i,k-1}$. However, these are precisely the vertices for which the algorithm adds a shortcut. □

B Omitted Content from Section 4

In this section, we investigate the geometric properties of the wavefront.

Lemma 13 *Any ray emanating at p_i intersects the wavefront at most once.*

Proof. We prove this statement inductively. As $W_{i,i+1} = D_{i,i+1}$, consider the wave of $D_{i,i+1}$. Since the unit circle in any L_p norm for $p \in [1, \infty]$ is convex, any ray emanating at p_i intersects a unit circle at most twice. The first intersection is with the bottom arc of the unit circle C_{i+1} and the second intersection is with the top arc of C_{i+1}. As the wave of $D_{i,i+1}$ is defined as the bottom arc of C_{i+1}, any ray emanating at p_i intersects the wave of $D_{i,i+1}$ at most once.

It remains to show the induction step for all $j > i+1$. By the induction hypothesis, we know that any ray emanating at p_i intersects the wavefront of $W_{i,j-1}$ at most once. The wavefront of $W_{i,j}$ is the boundary of the intersection of the valid region of $W_{i,j-1}$ and the local valid region of $D_{i,j}$. Consider a ray R originating at p_i. The ray R enters the valid region of $W_{i,j-1}$ at most at one point q where it also intersects the wavefront of $W_{i,j-1}$, and it enters the local valid region of $D_{i,j}$ at most at one point q' where it also intersects the wave of $D_{i,j}$. Hence, R enters the intersection of the valid region of $W_{i,j-1}$ and the local valid region of $D_{i,j}$ at most at one point – namely either at q or at q' (or $q = q'$). This is the only point of the wavefront of $W_{i,j}$ that is shared with R. □

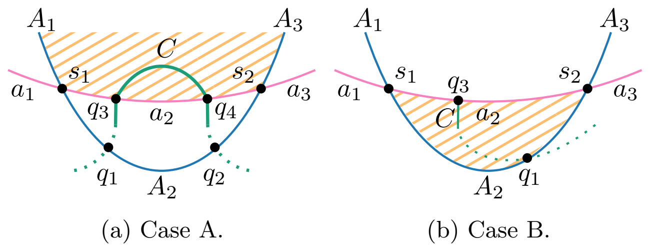


Figure 9: Cases in the proof of Lemma 4.

We can make a similar statement for unit circles. The number of intersections between a unit circle and the wavefront is relevant for updating the wavefront.

Lemma 4 (⋆) *Any unit circle of radius δ intersects the wavefront at most twice.*

Proof. We prove this statement inductively. Say p_i is our start vertex and we consider the wavefront of $W_{i,i+1}$, which is the same as the wave of $D_{i,i+1}$, which is part of the boundary of a unit circle. Since each pair of unit circles in the L_p norm for $p \in [1, \infty]$ intersects at most twice, we know that any unit circle intersects the wavefront of $W_{i,i+1}$ at most twice.

It remains to show the induction step for all $j > i+1$. Assume for a contradiction that a unit circle C intersects the wavefront of $W_{i,j}$ more than twice. Observe that the wavefront of $W_{i,j}$ is a subset of the wavefront of $W_{i,j-1}$ and the wave of $D_{i,j}$. Say C intersects the wavefront of $W_{i,j-1}$ at q_1 and q_2 and C intersects the wave of $D_{i,j}$ at q_3 and q_4 (maybe one of these points does not exist.) Next, we argue topologically that at most two points of $\{q_1, q_2, q_3, q_4\}$ lie on the wavefront of $W_{i,j}$, which is a contradiction.

By the induction hypothesis, the wavefront of $W_{i,j-1}$ and the wave of $D_{i,j}$ intersect at most twice. Let these intersection points from left to right be s_1 and s_2; see Fig. 9. Let the subdivisions of the wavefront of $W_{i,j-1}$ and the wave of $D_{i,j}$ induced by s_1 and s_2 be A_1, A_2, A_3 and a_1, a_2, a_3, respectively. Some of them may be empty. Clearly, the wavefront of $W_{i,j}$ is either A_1–a_2–A_3 or a_1–A_2–a_3. By Lemma 10, we know that it cannot be a_1–A_2–a_3, therefore, it is A_1–a_2–A_3.

Next, we analyze the intersection points q_3 and q_4 (maybe q_4 does not exist). Either one or two of them lies on a_2 as otherwise there are no more than two intersection points of C with the new wavefront.

Case A: The intersection points q_3 and q_4 lie on a_2; see Fig. 9a. As both intersection points are between the unit circle C and the wave of $D_{i,j}$, i.e., a bottom arc of another unit circle, we know by Lemma 9 that q_3 and q_4 are contained in the top arc of C. Thus, there is no ray R to the left of q_3 or to the right of q_4 originating at p_i and intersecting the arc of C between q_3 and q_4 as otherwise R would intersect the top arc of C twice. Therefore, the arc of C between q_3 and q_4 lies in the valid region (hatched orange in Fig. 9a) without reaching A_1 or A_3. When C passes through q_3 and q_4, it reaches the region between a_2 and A_2. If there are intersections between $W_{i,j-1}$ and C, they both lie on A_2.

Case B: Only one intersection point, let it be q_3, lies on a_2; see Fig. 9b. If it is a touching point, then C lies in the region between a_2 and A_2 before and after reaching q_3 (because we can assume that both unit circles are non-identical). If it is an intersection point, then C passes through q_3 into the region between a_2 and A_2 (hatched orange in Fig. 9b). To leave this region, q_1 (or q_2) lies on A_2. Hence, there are at most two points of $\{q_1, q_2, q_3, q_4\}$ on the new wavefront. □

Lemma 5 ($\star$) *The wavefront consists of at most $\mathcal{O}(n)$ arcs under any $L_{p\in(1,\infty)}$ norm.*

Proof. According to the inductive definition, we start with a wavefront consisting of one arc. Now in each step where we extend the wavefront, we consider the intersection between the current valid region and a local valid region – one is defined by the current wavefront, the other is defined by a single arc a. This is the intersection between the current wavefront and the unit circle on which a lies. By Lemma 4, we know that there are at most two intersection points. This means, the number of arcs on the wavefront increases by at most two. In the worst case, we start at vertex p_1 and adjust the wavefront $n-1$ times until we have created the wavefront of $W_{1,n}$. Therefore, any wavefront consists of at most $2n-3 \in \mathcal{O}(n)$ arcs[5]. □

Theorem 6 ($\star$) *An n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_2 (Euclidean) norm in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n)$ space.*

Proof. According to Lemmas 2 and 3, the algorithm we describe in Sec. 3 finds all valid shortcuts. It remains to analyze the runtime. We consider each of the n vertices as potential shortcut starting point p_i. When we encounter a vertex p_j with $j > i$, we determine in logarithmic time whether it is in the valid region. We do this by computing the ray emanating at p_i and going through p_j, and querying the arc it intersects in the wavefront. Then, using the case distinction of Lemma 14, we update the wavefront and the wedge. This needs amortized logarithmic time and over all steps $\mathcal{O}(n \log n)$ time.

Consequently, we construct the shortcut graph in $\mathcal{O}(n^2 \log n)$ time. In the resulting shortcut graph, we can find an optimal polyline simplification by finding a shortest path in $\mathcal{O}(n^2)$ time.

Regarding space consumption, we observe that the wavefront maintenance only requires linear space at any time. As we can compute the set of outgoing shortcuts of each vertex p_i individually, we can also easily apply the space reduction approach described for the Imai–Iri algorithm in Sec. 2 to get an overall space consumption in $\mathcal{O}(n)$. □

[5]One can further observe that, by Lemma 10, the number of arcs on the wavefront increases actually by at most one per vertex p_j ($j \in \{2, \ldots, n\}$). This means any wavefront consists of at most $n-1$ arcs.

Lemma 7 ($\star$) *In the L_∞ (L_1) norm, the wavefront consists of either one or two orthogonal line segments, which are horizontal or vertical (rotated by 45 degrees).*

Proof. We show this claim inductively. For $W_{i,i+1} = D_{i,i+1}$ it is just the bottom arc of a square (the unit circle in L_1 or L_∞). Clearly, this is either one or two orthogonal line segments – horizontal or vertical line segments in the L_∞ norm and line segments rotated by 45 degrees in the L_1 norm.

When we compute the wavefront of $W_{i,j}$, we compute the intersection of the valid region of $W_{i,j-1}$ (which is bounded by one or two orthogonal line segments by the induction hypothesis) and the local valid region of $D_{i,j}$ (which is bounded by one or two line segments parallel to the ones of $W_{i,j-1}$). Computing the boundary of this intersection in the L_∞ norm can be done by computing the intersection of two axis-parallel rectangles. The intersection of two axis-parallel rectangles is again an axis-parallel rectangle. In the L_1 norm, the situation is the same but rotated by 45 degrees. □

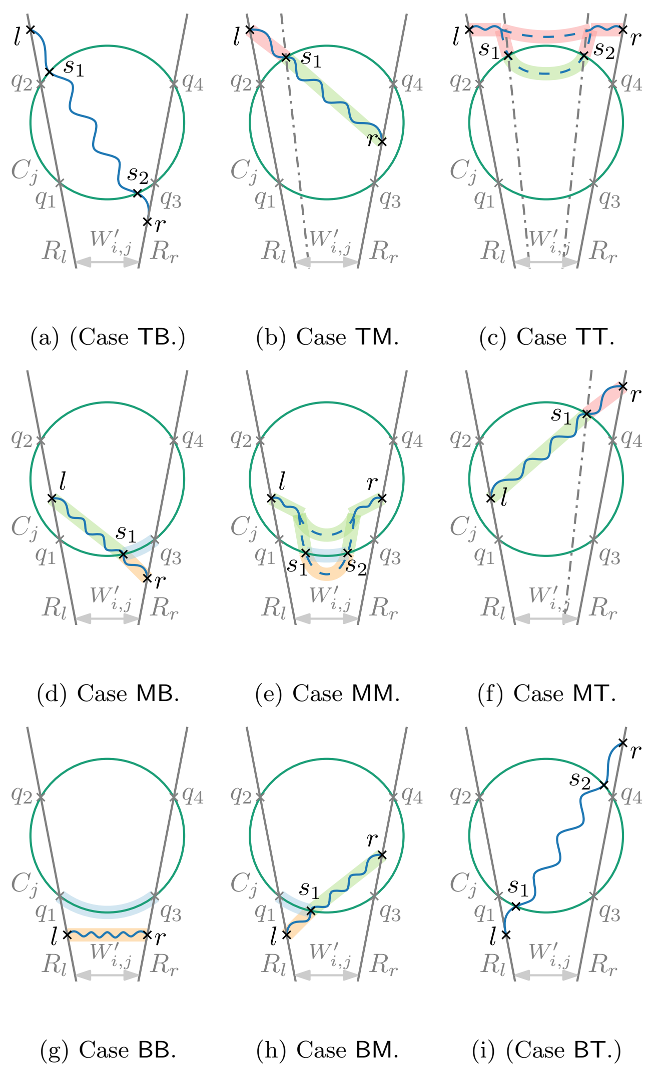
B.1 Procedure for Wavefront Maintenance in L_2

Now consider our algorithm under the L_2 norm, i.e., the Euclidean norm. In this section, we describe a procedure based on a case distinction to update the wavefront in amortized logarithmic time. For an overview, see Fig. 10.

Lemma 14 *Given a two-dimensional n-vertex polyline L and a vertex $p \in L$, we can find all valid shortcuts under the local Fréchet distance starting at p in $\mathcal{O}(n \log n)$ time.*

Proof. Say we are computing all valid shortcuts starting at p_i and we are currently processing a vertex p_j, which is the center of the unit circle C_j. We have already constructed the intermediate wedge $W'_{i,j}$ and clipped the wavefront of $W_{i,j-1}$ along the left and the right bounding rays R_l and R_r of $W'_{i,j}$. For this clipping, we may have removed a linear number of arcs, however, over all iterations we remove every arc at most once. Now, both R_l and R_r intersect C_j twice or touch C_j. Let q_1 and q_2 denote the intersection points between R_l and C_j (where q_1 is on the bottom arc of C_j). Similarly, let q_3 and q_4 denote the intersection points between R_r and C_j (where q_3 is on the bottom arc of C_j). Moreover, let l and r denote the intersection point between the wavefront and R_l and between the wavefront and R_r, respectively.

The relative positions of l, q_1, and q_2 on R_l and the relative positions of r, q_3, and q_4 on R_r determine where the intersection points s_1 and s_2 (if they exist) between C_j and the wavefront of $W_{i,j-1}$ can lie. (Recall that there are at



(a) (Case TB.) (b) Case TM. (c) Case TT.

(d) Case MB. (e) Case MM. (f) Case MT.

(g) Case BB. (h) Case BM. (i) (Case BT.)

Figure 10: Updating the wavefront (blue curve) when p_j is added. The red and orange parts are removed, while the green and light blue parts are the new wavefront.

most two intersection points by Lemma 4.) In the following, we write $a \prec b$ if a is below b along the ray R_l or R_r. If $q_1 = l$ or $q_2 = l$, then we proceed as if R_l was moved to the right by a tiny bit (symmetrically as if R_r was moved to the left). Thus, at such a point, the angle of the incident arc of the wavefront and C_j matters for the order. For the degenerate case $q_1 = l = q_2$, which includes a touching point between R_l and C_j, we hence assume $q_1 \prec l \prec q_2$.

Next, we consider all orderings of l, r, q_1, q_2, q_3, and q_4. This gives rise to the following nine cases. We remark that two of them (Case TB and Case BT) cannot occur and two pairs of the remaining cases are symmetric, which leaves essentially five different cases.

(Case TB:) $q_1 \prec q_2 \prec l$ and $r \prec q_3 \prec q_4$; see Fig. 10a. This case cannot occur. Suppose for a contradiction that we have this configuration. Then, there are precisely two intersection points s_1 and s_2 between the wavefront and C_j. The left intersection point s_1 is between the top arc of C_j and an arc a_k of the wavefront, which is part of the bottom arc of a unit circle C_k belonging to a vertex p_k with $i < k < j$. By Lemma 1, C_k contains the whole wavefront, thus including s_2, which means C_k and C_j intersect a second time such that this intersection point is to the left of s_2. Then, however, C_k and C_j intersect once with both bottom arcs and once with a bottom and a top arc – a contradiction to Lemma 9. For more details on this argument, see in Appendix A the proof of Lemma 1 and Fig. 8.

Case TM: $q_1 \prec q_2 \prec l$ and $q_3 \prec r \prec q_4$; see Fig. 10b. There is an intersection point s_1 between C_j and the wavefront. We traverse[6] the arcs in the balanced search tree representing the wavefront starting at the leftmost arc, which in turn starts at point l, and remove all arcs that we encounter until we find the intersection point s_1 between C_j and an arc a of the wavefront. We update a to start at s_1 and the left bounding ray of $W_{i,j}$ to go through s_1. There cannot be a second intersection point because otherwise there would also be a third intersection point between a unit circle and the wavefront.

Case TT: $q_1 \prec q_2 \prec l$ and $q_3 \prec q_4 \prec r$; see Fig. 10c. In this case, we either have two or no intersection points between the wavefront and C_j. We traverse the wavefront starting at l and remove all arcs that we encounter and that do not intersect C_j. If we do not find any intersection point but reach r, then there cannot be any further valid shortcut starting at v_i and we abort. Otherwise, we have found s_1 and proceed symmetrically at r to find s_2.

Case MB: $q_1 \prec l \prec q_2$ and $r \prec q_3 \prec q_4$; see Fig. 10d. There is precisely one intersection point s_1 between the wavefront and the bottom arc of C_j. We traverse the wavefront starting at r and remove all arcs that we encounter and that do not intersect C_j until we have found s_1. We clip the arc of the wavefront at s_1 and append the bottom arc of C_j between s_1 and r to the wavefront.

Case MM: $q_1 \prec l \prec q_2$ and $q_3 \prec r \prec q_4$; see Fig. 10e. There are either two or no intersection points between the wavefront and C_j. If there are two intersection points, then they are on the bottom arc of C_j as otherwise, it would contradict Lemma 10. The order of the arcs on the wavefront around p_i is reverse to the order of the corresponding unit circle centers around p_i [22]. By binary search, we determine the arcs a_k and a_{k+1} such that p_j lies in between the corresponding unit circle centers of a_k and a_{k+1} with respect to the angle around p_i. If a_k and a_{k+1} are completely contained inside C_j, then there is no intersection point and the wavefront remains unchanged.

Otherwise, we traverse the wavefront starting at a_k to the left until we have found an arc intersecting C_j, which gives us s_1. We remove all arcs along the way and split the arc containing s_1 at s_1. Symmetrically, we traverse the wavefront starting at a_{k+1} to the right to find s_2. Finally, at the resulting gap, we insert the arc of C_j between s_1 and s_2 into the wavefront.

Case MT: $q_1 \prec l \prec q_2$ and $q_3 \prec q_4 \prec r$; see Fig. 10f. This case is symmetric to Case TM.

[6]In the following we just say for short, "we traverse the wavefront starting at l".

Case BB: $l \prec q_1 \prec q_2$ and $r \prec q_3 \prec q_4$; see Fig. 10g. There is no intersection point between the wavefront and C_j. There cannot be a single intersection point and if there were two intersection points, it would contradict Lemma 10. We replace the whole wavefront by the arc of C_j from q_1 to q_3.

Case BM: $l \prec q_1 \prec q_2$ and $q_3 \prec r \prec q_4$; see Fig. 10h. This case is symmetric to Case MB.

(Case BT:) $l \prec q_1 \prec q_2$ and $q_3 \prec q_4 \prec r$; see Fig. 10i. Since this configuration is symmetric to Case TB, this case also cannot occur.

Note that we only do binary search in logarithmic time or if we traverse multiple arcs, we remove them from the wavefront. During the whole process, we add, for any vertex p_j with $j > i$, at most one arc to the wavefront. Therefore, we conclude the correctness of the lemma. □

C Omitted Content from Section 5

C.1 Implementing the Wavefront Data Structure

In Section 4 we based our asymptotic running time analysis on the assumption that the wavefront data structure is most efficiently implemented by a balanced search tree (e. g., a red-black tree). While this is reasonable for wavefronts in the size of $\mathcal{O}(n)$, we learned in Section 5 that wavefronts are consistently small on real-world input data (see also Fig. 3b).

This raises questions about the appropriate data structures, and the best search strategy (binary, linear), for small wavefronts. We conducted experiments on several implementation variants, using the input data set from Section 5. The red-black tree implementation is based on a library by Lukas Barth [6]. Linked lists use a Boost library by Krzikalla and Gaztanaga [21]. All other variants are made by ourselves. Results are shown in Fig. 11.

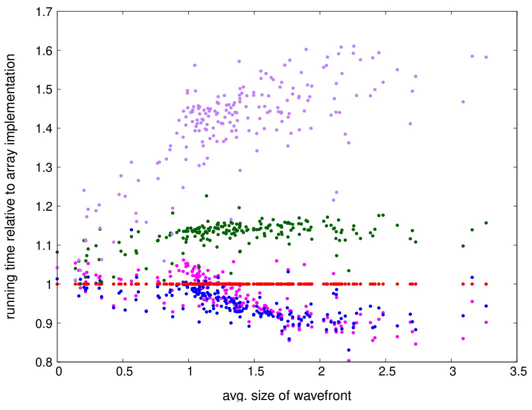
We draw the following conclusions:

(1) Most notably, linear search (blue, pink) is superior to binary search (red, green, purple) on small wavefronts. The left-right decisions inherent to binary search do not come for free. Linear search does not need these decisions, turning its (asymptotic) disadvantage into an advantage.

(2) Arrays (red) are superior to trees (green). This may come as a surprise, since the update costs of arrays are in $\mathcal{O}(n)$. We note, however, that updates at the beginning and end of the array can be done in $\mathcal{O}(1)$, using a double-ended array structure. Updates to the center of the array still involve moving $\mathcal{O}(n)$ elements (a contiguous block of pointers, actually). Such copy operations are, fortunately, performed in a tight loop that can be optimized very well on modern hardware (e. g, using vectorized instructions). In contrast, updates to a balanced tree, despite being in $\mathcal{O}(\log n)$, require branching conditions that are less well-suited for optimization.

(3) For similar reasons, arrays (blue) are competitive to linked lists (pink). Incidentally, there is very little difference in performance between a singly-linked list and a doubly-linked lists. This may be explained by the fact that a singly-linked list needs to maintain an additional pointer to the end of the list (while the doubly-linked list uses a circular structure).

(4) Skip lists (purple) lose to red-black trees (green), despite having very similar asymptotic characteristics. We attribute this to larger constant factors in the implementation. Skip list maintenance is, we assume, more expensive and/or less cache-friendly than balancing a tree.



data structure	search	plot series
skip list	binary	purple
red-black tree	binary	green
array	binary	red baseline = 1
array	linear	blue
linked list	linear	pink

Figure 11: Comparison of running times between different implementation variants.

Partition, Reduction, and Conquer: A Geometric Feature-Based Approach to Convex Hull Computation

Kyuseo Park *   Markus Schneider†

Abstract

This paper introduces the Partition, Reduction, and Conquer Convex Hull (PRC-CH) algorithm, a novel geometric feature-based approach for computing planar point set convex hulls. Inspired by the Akl-Toussaint heuristic, the PRC-CH algorithm iteratively identifies candidate points by leveraging their geometric properties and discarding non-convex hull points. With a worst-case time complexity of $O(n^2)$, the PRC-CH algorithm's efficiency is assessed through comparison with well-known convex hull algorithms in square-boundary, disk-boundary, and circular boundary scenarios. The PRC-CH algorithm proves competitive, outperforming others in most cases except for extreme point distributions. It offers a fresh perspective on the convex hull problem and holds potential for further improvement by refining elimination strategies and addressing limitations in handling extreme point distributions.

1 Introduction

The convex hull of a set of points is the smallest convex set that encompasses the entire point set. Computing the convex hull of a finite set of points in a plane is considered one of the fundamental problems in computational geometry. Convex hull algorithms find applications in a wide array of fields, including Geographic Information Systems (GIS), computer graphics, and robotics. For example, in the context of a contagious disease outbreak like the recent COVID-19 pandemic, estimating the spatial extent of an epidemic can be achieved by computing the convex hull. Furthermore, computing the convex hull is an essential preliminary step in determining the diameter of a point set. Consequently, numerous techniques have been developed to expedite computational results and enhance the efficiency of the convex hull computation process.

The Akl-Toussaint heuristic is a technique that improves the efficiency of convex hull algorithms by identifying the four extremal points that form the leftmost, bottommost, rightmost, and topmost points of a point set. This heuristic constructs a polygon using these four points and eliminates all points inside the polygon, as they are not part of the final convex hull. This approach substantially reduces the number of points requiring processing in subsequent computation steps. By eliminating interior points using the four extremal points, the Akl-Toussaint heuristic optimizes the convex hull algorithm's computation process, leading to more efficient performance.

Moreover, the heuristic offers the advantage of identifying the four extremal points that belong to the final convex hull in advance. These points can be leveraged in later steps to help identify potential final convex hull vertices. As a result, researchers [1, 3, 5, 8] have explored convex hull algorithms that incorporate the Akl-Toussaint heuristic as a preprocessing step to take advantage of its benefits.

Although the Akl-Toussaint heuristic can provide additional benefits, particularly in offering geometric features and locality principles for points outside the polygon, previous research has only superficially utilized these advantages. The key distinction between the PRC-CH algorithm and existing algorithms that employ the Akl-Toussaint heuristic lies in the way it identifies and processes candidate points for the convex hull. By systematically evaluating the geometric properties of points, the algorithm can eliminate non-candidate points and subsequently compute the final convex hull using the remaining candidate points. This approach enhances the algorithm's efficiency, making it competitive with other well-established algorithms.

This paper is organized as follows. In Section 2, we provide an overview of related work, focusing on convex hull algorithms that incorporate the Akl-Toussaint heuristic. Our novel convex hull algorithm, which leverages geometric properties and locality principles of points to compute the final convex hull, is described in detail in Section 3. Section 4 offers an analysis of our algorithm's complexity. Experimental results and comparisons with other algorithms are presented in Section 5. Finally, we outline our conclusions and directions for future research in Section 6.

*Computer & Information Science & Engineering, University of Florida, kyuseopark@ufl.edu

†Computer & Information Science & Engineering, University of Florida, mschneid@cise.ufl.edu

2 Akl-Toussaint Heuristic and Its Application in Convex Hull Algorithms

Akl and Toussaint introduced a convex hull algorithm and proposed a new method as part of their research [1], known as the Akl–Toussaint heuristic. This method comprises two stages. The first stage involves identifying the four extremal points among the initial point set, which are identified by their minimum and maximum x- and y-coordinates, respectively, as shown in Figure 1. These four points must belong to the final convex hull. In the second stage, points that are inside the polygon built by the four extremal points are discarded, ensuring that none of the points belongs to the convex hull. Although the Akl–Toussaint heuristic consists of only two simple steps, it offers crucial benefits. Firstly, by discarding a certain number of points, algorithms that use this heuristic as a preprocessing step can achieve performance improvements. Secondly, the four extremal points, which are used to form the initial polygon, serve as the foundation for convex hull algorithms. As a result of its benefits, the Akl-Toussaint heuristic has been widely adopted as a preprocessing step in many convex hull algorithms.

Akl and Toussaint's convex hull algorithm [1] The algorithm initially identifies four extremal points and subsequently eliminates all the points falling inside the polygon, which is constructed by connecting these extremal points. Subsequently, the algorithm partitions the remaining point set into four distinct regions, which are created by the sides of the polygon and the corner of the bounding box of the polygon. For each region, the algorithm sorts the points based on their x-coordinates and proceeds to determine the vertices of the convex hull for that particular region. This is achieved by examining every three consecutive points, and identifying whether they form a counterclockwise turn or not. In case they form a counterclockwise turn, the algorithm continues to analyze the next three consecutive points. However, if they do not form a counterclockwise turn, the algorithm skips the middle point and includes the point that succeeds the last point of the three.

Several convex hull algorithms have utilized the Akl-Toussaint heuristic, but they do not fully exploit its potential.

Bhattacharya and Toussaint's algorithm [3] is similar to Akl and Toussaint's method. It identifies four extremal points, eliminates points within the formed polygon, sorts the remaining points, and then constructs a monotone polygon to compute the convex hull.

Mei, Tripper, and Xu's algorithm [8] involves identifying extreme points, sorting the remaining points, recursively creating *e-Quads*, assembling a simple polygon, and computing the convex hull.

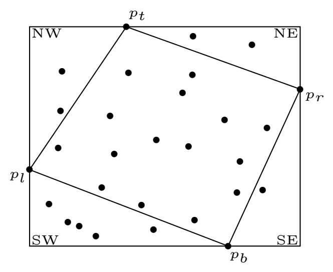


Figure 1: Illustration of the Akl-Toussaint heuristic. The four extremal points p_l, p_r, p_t, and p_b are identified based on their minimum and maximum x- and y-coordinates. The initial polygon built by these extremal points divides the plane into four zones: SW (South West), SE (South East), NE (North East), and NW (North West).

Fu and Lu's algorithm [5] sorts points, discards points inside a polygon formed by extreme points, divides the remaining points into sub-regions, and deletes concave points on the initial boundary to obtain the final convex hull vertices.

All these algorithms that employ the Akl-Toussaint heuristic have a time complexity of $O(n \log n)$, primarily attributed to the sorting process they include. This sorting step creates an inherent performance limitation, constraining the potential for further optimization in terms of computation time.

While these algorithms focus on utilizing heuristics to decrease the number of points requiring further processing and using extremal points to divide a given set of points into subsets or as a basis for calculating the final convex hull vertices, they do not fully exploit the advantages offered by the Akl-Toussaint heuristic. There exist other significant benefits, such as obtaining locality principles of points, which are overlooked by existing algorithms. In this regard, there is potential for developing new approaches that more effectively harness the power of the Akl-Toussaint heuristic to optimize the convex hull computation process.

3 Partition, Reduction, and Conquer Convex Hull Algorithm Description

The proposed algorithm aims to reduce the computational burden of various operations, including sorting and convex hull vertex identification, by fully utilizing the geometric features of the points. It comprises three main steps: spatially partitioning the point set into subsets, eliminating non-candidate points, and computing local convex hulls from the remaining candidate points and combining them to form the global convex hull. Each step is detailed in the following subsections.

3.1 Spatial Partitioning and Locality Principles of Points

The algorithm first identifies the leftmost, rightmost, topmost, and bottommost extremal points and eliminates non-candidate points within the polygon formed by the four extremal points. The algorithm then identifies distinct, non-overlapping sub-regions, referred to as *zones* in our proposed algorithm: SW (South West), SE (South East), NE (North East), and NW (North West), based on the four extremal points, as shown in Figure 1. Each remaining point is allocated to a specific zone based on its spatial location relative to the extremal points. As a result, points in each zone possess similar spatial characteristics that lead to unique locality properties specific to the zone. Let P_{SW} (P_{SE}, P_{NE}, P_{NW}) denote the set of points located in the SW (SE, NE, NW) zone, and let p_l, p_b, p_r, and p_t represent the leftmost, bottommost, rightmost, and topmost points, respectively. Points in each zone can be described by a set of locality principles as follows:

$$\begin{aligned} P_{SW} =&\{p(x,y) \mid p_l.x < x < p_b.x \wedge \ p_b.y < y < p_l.y, \\ &\text{and } p \text{ lies to the right of the vector } \overrightarrow{p_l p_b}\} \end{aligned}$$

$$\begin{aligned} P_{SE} =&\{p(x,y) \mid p_b.x < x < p_r.x \wedge \ p_b.y < y < p_r.y, \\ &\text{and } p \text{ lies to the right of the vector } \overrightarrow{p_b p_r}\} \end{aligned}$$

$$\begin{aligned} P_{NE} =&\{p(x,y) \mid p_t.x < x < p_r.x \wedge \ p_r.y < y < p_t.y, \\ &\text{and } p \text{ lies to the right of the vector } \overrightarrow{p_r p_t}\} \end{aligned}$$

$$\begin{aligned} P_{NW} =&\{p(x,y) \mid p_l.x < x < p_t.x \wedge \ p_l.y < y < p_t.y, \\ &\text{and } p \text{ lies to the right of the vector } \overrightarrow{p_t p_l}\} \end{aligned}$$

The locality principles for each zone are determined by the position of points relative to the directed segment formed by the two extremal points that define the zone. For instance, in the SW zone, points are located within the bounding box formed by the segment $\overrightarrow{p_l p_b}$ and are required to lie to the right of the segment.

Lemma 1 *Points in each zone are candidates for the convex hull.* [1]

Furthermore, the spatial partitioning method and the locality principles for points within each zone guarantee that each point belongs to a single subset. This avoids overlapping subsets and allows the algorithm to efficiently handle point subsets based on their respective zones, without iterating over the entire point set.

While our focus is on computing the local convex hull in the SW zone, the convex hulls in the other three zones follow the same symmetric logic. Cases with fewer than four zones can be handled during implementation and are not discussed here.

[1]Proof of this lemma is provided in Appendix A.

3.2 Reduction Process in each Zone

Lemma 1 establishes that all points in each zone are potential candidates for inclusion in the final convex hull. However, not all these points will necessarily be part of the final convex hull. Therefore, after applying the Akl-Toussaint heuristic, various algorithms utilize specific methods to identify the vertices of the convex hull. One common approach is to sort points as a preprocessing step, as discussed in Section 2. After sorting points, those that are unlikely to be the final convex hull vertices are eliminated during the computation of the convex hull. Our proposed algorithm offers an alternative approach for computing the convex hull vertices, without the need for a general sorting process.

Points that fail to satisfy their zone's locality principles are considered non-candidates for the final convex hull and should be discarded. The proposed algorithm aims to identify new extremal points within each zone, enabling the redefinition of locality principles and eliminating as many non-candidate points as possible. Once these new extremal points are found, locality principles are redefined accordingly, and points that do not satisfy the updated principles can be discarded.

In the case of the SW zone, the algorithm identifies two points that have equivalent characteristics to the extremal points p_l and p_b, designating them as p_{l1} and p_{b1}, respectively. p_{l1} is the point with the smallest x-coordinate, while p_{b1} is the point with the smallest y-coordinate among points in the SW zone. By replacing the original extremal points with these two new extremal points, the algorithm redefines the locality principles for the SW zone. Let RP_{SW} represent the set of remaining points that satisfy the locality principles of the SW zone with these replacements. The redefined locality principles for the remaining points in the SW zone can then be expressed as follows:

$$\begin{aligned} RP_{SW} =&\{p(x,y) \mid \exists i \in 1,\dots,m : \\ &p_{li}.x < x < p_{bi}.x \wedge \ p_{bi}.y < y < p_{li}.y, \\ &\text{and } p \text{ lies to the right of the vector } \overrightarrow{p_{li} p_{bi}}, \\ &m : \text{the final identification of extremal points}\} \end{aligned}$$

p_{li} and p_{bi} are the extremal points in the set of points being processed. Points that lie to the right of the segment $\overrightarrow{p_{li} p_{bi}}$ in the SW zone satisfy the newly defined locality principles. Non-candidate points located to the left of the segment are not considered as vertices of the final convex hull and are therefore eliminated. We refer to the process of eliminating non-candidate points as the "*Reduction*" process. This involves identifying two new extremal points in each zone with the same characteristics as the original two extremal points and eliminating the points that lie to the left of the segment connecting these new extremal points. The segment utilized for this process is referred to as the "*reduction segment*".

Lemma 2 *After the reduction process in each zone, remaining points are candidates for the convex hull.* [2]

The reduction process is performed iteratively, and in each iteration, the locality principles of each zone are updated using the newly found extremal points. Based on these updated locality principles, points that do not satisfy the revised criteria are eliminated from the zone.

In each iteration of the reduction process, newly identified extremal points are regarded as confirmed candidates for the final convex hull. These "*confirmed candidates*" are more likely to be part of the final convex hull compared to other candidate points, which may be eliminated in subsequent iterations. The algorithm stores these confirmed candidates throughout the process. As the algorithm proceeds, the stored extremal points from each iteration will eventually form the set of confirmed candidate points for the final convex hull. Figure 2 demonstrates the reduction process in the SW zone, illustrating the identification of extremal points, the updating of the locality principle, and the formation of two sets of confirmed candidates.

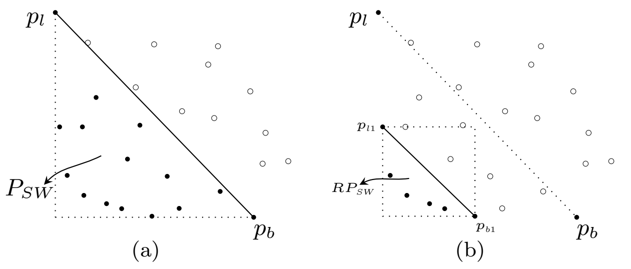


Figure 2: Examples of the reduction process in the SW zone: (a) Points in the SW zone with initial extremal points p_l and p_b defining the locality principle. (b) Updated locality principle using newly found extremal points p_{l1} and p_{b1}, eliminating non-satisfying points, and forming two sets of confirmed candidates: one with p_l and p_{l1}, and another with p_b and p_{b1}.

3.3 Computation and Merging of Local Convex Hulls in the Conquer Phase

During the reduction process, two distinct sets of confirmed candidate points are generated for each zone, which are used to compute the local convex hull. In the SW zone, one set contains points with the smallest x-coordinates, while the other set consists of points with the smallest y-coordinates. These points contribute to the local convex hull computation. The inherent geometric properties of each set are leveraged to construct the local convex hull.

[2]Proof of this lemma is provided in Appendix A.

Lemma 3 *Points in both sets of confirmed candidates within each zone are already sorted in order of their x-coordinates.* [3]

Zone	Set of	Pattern
SW	the smallest x-coordinates points	increasing
	the smallest y-coordinates points	decreasing
SE	the largest x-coordinates points	decreasing
	the smallest y-coordinates points	increasing
NE	the largest x-coordinates points	decreasing
	the largest y-coordinates points	decreasing
NW	the smallest x-coordinates points	increasing
	the largest y-coordinates points	decreasing

Table 1: Summary of x-coordinate patterns for each zone's point sets.

Lemma 4 *Points in the two sets of confirmed candidate points within each zone do not intersect.* [4]

Lemma 3 demonstrates that the locality principles guarantee the order of the points' x-coordinates in each set for every zone, and the geometric properties of the confirmed candidate points in each zone can be concisely represented in Table 1. Furthermore, *Lemma* 3 and *Lemma* 4 demonstrate that the two sets of confirmed candidate points for each zone can be combined to produce a set of points arranged by their x-coordinates. These ordered point sets from each zone are employed to create the local convex hull for each respective zone using the following method: three consecutive points are evaluated to identify counterclockwise turns. If such turns are detected, the algorithm advances to the next triplet; otherwise, the middle point is omitted, and the analysis proceeds with the subsequent point. After constructing the local convex hulls for each zone, they are merged to form the final convex hull that encompasses the entire point set.

4 Complexity Analysis

In this section, we analyze the computational complexity of the proposed algorithm. We discuss each step's complexity, determine the overall time complexity, and also present the space complexity.

4.1 Partition of Points into Zones

The initial step involves partitioning the input points into four zones, requiring the determination of extremal points and assigning each point to a zone based on its position relative to these extremal points. Both processes have a time complexity of $O(n)$, where n is the

[3]Proof of this lemma is provided in Appendix A.
[4]Proof of this lemma is provided in Appendix A.

number of input points. Thus, the complexity of this step is $O(n)$.

4.2 Reduction Process

The complexity of the reduction process depends on the number of iterations. Each iteration identifies two new extremal points and eliminates points that do not satisfy the locality principles. The worst-case time complexity occurs when all input points, excluding the extremal points, must be visited in each iteration, resulting in a complexity of $O(n^2)$.

4.3 Computation and Merging of Local Convex Hulls

Computing local convex hulls involves using a set of points sorted by their x-coordinates, with a time complexity of $O(n)$. Merging the local convex hulls has a complexity of $O(1)$. Consequently, the overall complexity of this step is $O(n)$.

4.4 Overall Time Complexity

The overall complexity of the proposed algorithm is determined by summing the complexities of each step. With n referring to the number of input points, the worst-case complexity is $O(n^2)$, while the average-case complexity is expected to be lower, depending on the reduction process efficiency and input point distribution. [5]

4.5 Space Complexity

As the proposed algorithm iteratively processes the input points, it maintains a list of remaining points while discarding the non-candidates from further consideration. Since the algorithm does not require any additional data structures for storing intermediate results or complex transformations, its space complexity is determined by the number of input points. Therefore, the space complexity of the algorithm is $O(n)$, where n represents the total number of points in the input set.

5 Experimental Results

We conducted a performance evaluation of our proposed algorithm by comparing it to well-known convex hull algorithms, such as the Graham Scan algorithm [6], Quickhull algorithm [2, 4], and Akl-Toussaint's algorithm [1]. The experiments were conducted on a system equipped with an Apple *M1* processor and 16 GB of memory. To generate input data, we created two sets of points: one set within the shape of a square and the other within a disk. The number of input points in each set ranged from 10,000 to 10,000,000. We generated 20 samples for each set of input points and determined the final results for each algorithm by averaging the outcomes of the 20 computations.

[5]The average-case time complexity is presented in Appendix C.

Algorithm	Size of Points			
	10^4	10^5	10^6	10^7
Quickhull	0.41	3.29	31.65	314.70
Graham Scan	1.56	16.96	195.25	2316.03
Akl-Toussaint	0.74	7.44	84.12	945.13
PRC-CH	0.32	2.51	25.41	239.47

Table 2: Computation times (in milliseconds) for convex hull algorithms when points are distributed randomly in a square-boundary.

Algorithm	Size of Points			
	10^4	10^5	10^6	10^7
Quickhull	0.38	3.43	33.51	378.70
Graham Scan	1.47	16.86	193.17	2295.44
Akl-Toussaint	0.35	3.50	38.55	434.55
PRC-CH	0.26	2.75	44.53	999.47

Table 3: Computation times (in milliseconds) for convex hull algorithms when points are distributed randomly in a disk-boundary.

Algorithm	Size of Points	
	10^4	10^5
Quickhull	4.29	46.01
Graham Scan	1.29	15.32
Akl-Toussaint	1.15	13.00
PRC-CH	20.07	1786.81

Table 4: Computation times (in milliseconds) for convex hull algorithms when points are distributed randomly on a circular boundary.

The selection of convex hull algorithms for comparison is based on the following rationale. The Graham Scan algorithm includes a sorting process as a preprocessing step, allowing us to analyze the impact of sorting on performance. The Akl-Toussaint algorithm uses the Akl-Toussaint heuristic and a sorting process, enabling us to evaluate our method's effectiveness in identifying confirmed candidate points and assess its ability to reduce the computational burden of sorting. We also compare our algorithm to the Quickhull algorithm, which has a worst-case time complexity of $O(n^2)$, similar to our method. This comparison demonstrates the performance capabilities of our proposed method. The Quickhull algorithm used in our comparison is a modified version minimizing the number of orientation tests, as introduced by Hoàng and Linh [7].

Based on the experimental results, our proposed PRC-CH algorithm demonstrates competitive performance in comparison to other convex hull algorithms in terms of computation time. In the square-boundary scenario (Table 2), the PRC-CH algorithm consistently outperforms the Quickhull, Graham Scan, and Akl-Toussaint algorithms across all input sizes. The PRC-CH algorithm achieves computation time reductions ranging from approximately 20% to 24% compared to the Quickhull algorithm. These results suggest that our proposed algorithm's strategy for eliminating non-convex hull vertices is more effective than the Quickhull algorithm when points are not generated in extreme cases, despite both algorithms sharing the same time complexity of $O(n^2)$. In comparison to the Graham Scan and Akl-Toussaint algorithms, the reductions in computation time range from approximately 79% to 86% and 56% to 66%, respectively, indicating that the sorting process before computing the convex hull adds significant computational overhead.

In the disk-boundary scenario (Table 3), the PRC-CH algorithm faces a higher computational burden due to the point distribution that hinders our reduction strategy's ability to eliminate more points. Nonetheless, it continues to display competitive results. The PRC-CH algorithm outperforms the Graham Scan algorithm across all input sizes, achieving computation time reductions of approximately 60% to 88%. Although the PRC-CH algorithm's computation time exceeds that of the Akl-Toussaint algorithm for larger input sizes (1,000,000 and 10,000,000), it still maintains a competitive edge for smaller input sizes (10,000 and 100,000), with reductions of about 21% and 24%, respectively. The PRC-CH algorithm's performance is relatively close to the Quickhull algorithm for smaller input sizes but exhibits a higher computation time for larger input sizes.

When points are distributed in a circle, more points lie along the boundary, likely belonging to the convex hull vertices. This can lead to fewer eliminations and more iterations in the reduction process, reducing our algorithm's performance, especially when all input points lie on a circular boundary.

Table 4 displays the computation time results for circular point distributions where every point is a convex hull vertex, highlighting a weakness in our algorithm for such extreme cases. Algorithms incorporating sorting processes, like Graham Scan and Akl-Toussaint, demonstrate strengths, while those that do not, such as Quickhull and our proposed method, experience substantial computational burdens. During each iteration of the reduction process, our algorithm eliminates only two points, resulting in a set of $n-2$ points for the subsequent iteration. The Quickhull algorithm, on the other hand, processes a set of $n/2$ points for the following recursion, leading to better performance under such distributions. These distinctions in point elimination contribute to the differences in performance between the Quickhull and our proposed algorithm under extreme point distributions.

In summary, the experimental results suggest that our proposed PRC-CH algorithm is a promising candidate for computing convex hulls in various scenarios. Its performance is particularly noteworthy in the square-boundary scenario, where it consistently outperforms other established algorithms. In the disk-boundary scenario, the PRC-CH algorithm remains competitive, although it faces a higher computational burden. However, when points are distributed along a circular boundary, our algorithm exhibits a weakness, as it struggles with extreme point distributions. In such cases, algorithms with a sorting process display better performance.

6 Conclusion

We propose a novel convex hull algorithm, named Partition, Reduction, and Conquer Convex Hull algorithm (PRC-CH), which aims to compute the convex hull of a given set of points using a new concept. The proposed algorithm is designed based on the Akl-Toussaint heuristic and leverages geometric features to compute convex hull vertices while the algorithm runs.

The worst-case time complexity of the PRC-CH algorithm is $O(n^2)$. However, the average-case complexity is expected to be lower, depending on the efficiency of the reduction process and the distribution of input points. To evaluate the performance of our algorithm, we conducted experiments using randomly distributed input points within a square and a disk boundary and on a circular boundary. Our proposed PRC-CH algorithm demonstrates a competitive performance compared to other convex hull algorithms, with notable reductions in computation time, especially in the square-boundary scenario where the reductions range from 20% to 24% compared to the Quickhull algorithm. Despite the increased computational burden in the disk-boundary scenario, the PRC-CH algorithm still exhibits competitive results, outperforming the Graham Scan algorithm across all input sizes. However, our algorithm shows a weakness with extreme point distributions, such as those along a circular boundary.

These findings suggest that the PRC-CH algorithm is particularly effective in certain situations, but further research could investigate additional strategies for point elimination and distribution to enhance the algorithm's overall performance. This could involve exploring hybrid approaches that combine the strengths of different convex hull algorithms or optimizing the PRC-CH algorithm to address its weaknesses in handling extreme point distributions. By addressing these challenges, the PRC-CH algorithm has the potential to become an even more effective solution for a wider range of convex hull computation scenarios.

References

[1] S. G. Akl and G. T. Toussaint. A fast convex hull algorithm. *Information Processing Letters*, 7(5):219–222, 1978. https://doi.org/10.1016/0020-0190(78)90003-0

[2] A. Bykat. Convex Hull of a Finite Set of Points in Two Dimensions. *Information Processing Letters*, 7(6):296–298, 1978. https://doi.org/10.1016/0020-0190(78)90021-2

[3] B. K. Bhattacharya and G. T. Toussaint. Time- and storage-efficient implementation of an optimal planar convex hull algorithm. *Image and Vision Computing*, 1(3):140–144, 1983. https://doi.org/10.1016/0262-8856(83)90065-3

[4] W. F. Eddy. A New Convex Hull Algorithm for Planar Sets. *ACM Transactions on Mathematical Software*, 3(4):398–403, 1977. https://doi.org/10.1145/355759.355766

[5] Z. Fu and Y. Lu. An Efficient Algorithm for the Convex Hull of Planar Scattered Point Set. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(2):63–66, 2012. https://doi.org/10.5194/isprsarchives-XXXIX-B2-63-2012

[6] R. L. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133, 1972. https://doi.org/10.1016/0020-0190(72)90045-2

[7] N. Hoàng and N. K. Linh. Quicker than Quickhull. *Vietnam Journal of Mathematics*, 43:57–70, 2015. https://doi.org/10.1007/s10013-014-0067-1

[8] G. Mei, J. C. Tipper, and N. Xu. An Algorithm for Finding Convex Hulls of Planar Point Sets. In *Proceedings of the International Conference on Computer Science and Network Technology (ICCSNT)*, pages 888–891, 2012. https://doi.org/10.1109/ICCSNT.2012.6526070

[9] R. L. Graham, D. E. Knuth, and O. Patashnik. Concrete Mathematics: A Foundation for Computer Science. 2nd edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[10] R. P. Boas and J. W. Wrench. Partial Sums of the Harmonic Series. *The American Mathematical Monthly*, 78(8):864–870, 1971. http://www.jstor.org/stable/2316476.

Appendix

A Proofs

Proof. [Proof of Lemma 1] Consider a point $p(x, y)$ within the SW zone. Given that the point p is within the bounding box defined by the leftmost point p_l and the bottommost point p_b, p cannot exist beyond the boundaries of the bounding box. Also, any point located on the left side of the segment $\overrightarrow{p_l p_b}$ would contribute to a non-convex curve and thus would not be eligible to be a part of the convex hull.

Therefore, the potential candidates for the convex hull of the SW zone are the points that lie within the bounding box and to the right of the directed segment $\overrightarrow{p_l p_b}$. By definition of the locality principles for points in the SW zone, all points within the zone satisfy these criteria, and thus are appropriate candidates for inclusion in the convex hull. By applying similar reasoning to the other zones, we can conclude that the points within each zone are eligible candidates for consideration in the convex hull. □

Proof. [Proof of Lemma 2] Assume that there exists a point in a zone that violates the locality principles of the zone, yet has not been eliminated by the reduction process. This contradicts the assumption that the reduction process has been completed and all non-candidate points have been eliminated. Therefore, after the reduction process, only the points that satisfy the locality principles for each zone remain, and these points are candidates for the convex hull. □

Proof. [Proof of Lemma 3] Let us consider the SW zone with two sets of confirmed candidate points, set A and set B, generated during the iterative reduction process of the algorithm for the SW zone. Set A consists of points with the smallest x-coordinates, while set B consists of points with the smallest y-coordinates, both found during each iteration.

During each iteration, the reduction process identifies two new extremal points, one for set A and one for set B. Each new extremal point must be found within the bounding box formed by the previously identified extremal points. The locality principles ensure that the bounding box contains only points that have not yet been considered as confirmed candidates, and all points within the bounding box satisfy the locality principles.

When a new extremal point is identified for set A, it has a smaller x-coordinate than any other remaining point in the bounding box. Since the bounding box is formed based on the previously identified extremal points and their locality principles, it guarantees that any new extremal point added to set A has a larger x-coordinate than the previously identified points in set A. Thus, the points in set A are sorted in increasing order of their x-coordinates.

Similarly, when a new extremal point is identified for set B, it has a smaller y-coordinate than any other remaining point in the bounding box. As the bounding box is formed based on the previously identified extremal points and their locality principles, it guarantees that any new extremal point added to set B has a smaller x-coordinate than the previously identified points in set B. Thus, the points in set B are sorted in decreasing order of their x-coordinates.

Hence, we can conclude that the points in both sets A and B are sorted in x-coordinates order as a result of the reduction process and the locality principles.

□

Proof. [Proof of Lemma 4] We will consider the SW zone as a representative case. Let set A consist of confirmed candidate points with the smallest x-coordinates, and let set B consist of confirmed candidate points with the smallest y-coordinates. According to the locality principles of the SW zone, points in set A must be to the left of the points in set B. This is ensured through the reduction process, which generates a bounding box for each new extremal point found.

The locality principles and reduction process continue to apply until no more bounding boxes can be generated. As a result, the points in set A will always be to the left of the points in set B, ensuring that the two sets do not intersect.

The cases for the SE, NE, and NW zones can be similarly proven due to the symmetry of the problem. The geometric properties and locality principles in these zones are analogous to the SW zone, with adjustments made according to the specific extremal points and coordinate requirements of each zone. By applying the same reasoning as in the SW zone case, we can show that the two sets of confirmed candidate points in each of the other zones also do not intersect.

□

B Improving PRC-CH Algorithm: Supplemental Information

B.1 Selecting the Optimal Reduction Segment

At the n^{th} iteration of the reduction process, a discarded point is visited $2n$ times throughout the entire reduction process. This implies that the point is visited n times when the algorithm identifies the two new extremal points and n times when the algorithm eliminates points that do not satisfy the locality principles. Consequently, by discarding as many points as possible at each iteration, the algorithm can reduce the number of visits to each point, thus decreasing the overall number of operations. Identifying the optimal reduction segment is crucial for maximizing the number of points discarded at each iteration, which improves the efficiency and computational performance of the algorithm.

A reduction segment formed by two extremal points in a zone is not always optimal due to the possible positions of these extremal points. For instance, when connecting reduction segments with one of the endpoints of the reduction segment used in the previous iteration, a concave curve may be generated. Figures 3b and 3e illustrate scenarios in which the concave curves formed by p_l, p_{l1}, and p_{b1} and by p_b, p_{b1}, and p_{l1} are displayed, respectively. In these cases, the reduction segment is considered suboptimal because it does not eliminate a larger number of points, requiring additional iterations and potentially affecting the algorithm's efficiency. Consequently, to construct the optimal reduction segments, the proposed algorithm considers the convexity of the extremal points relative to those identified in previous iterations, as demonstrated in Figure 3c and Figure 3f. Extremal points removed from consideration for forming optimal reduction segments are also no longer regarded as confirmed candidates.

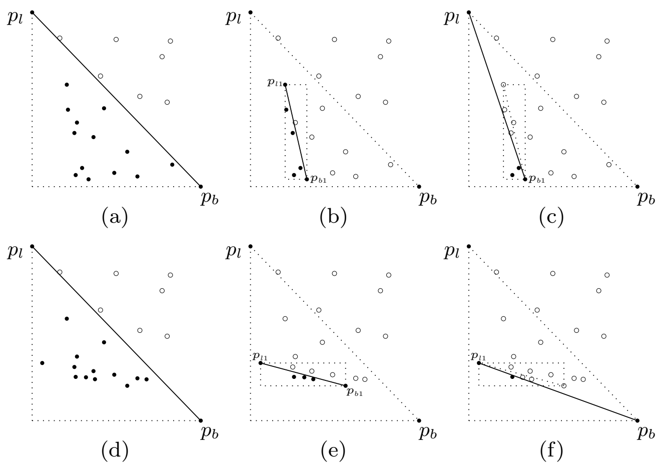


Figure 3: Examples of the suboptimal reduction segments and the optimal reduction segments are shown for different point distributions. (a) and (d) show examples of point distributions in the SW zone, respectively. The suboptimal reduction segments formed by p_{l1} and p_{b1} are shown in (b) and (e), while the optimal reduction segment formed by p_l and p_{b1} is shown in (c), and the optimal reduction segment formed by p_{l1} and p_b is shown in (f).

B.2 Reducing Computational Burden with Locality Principles

In addition to the construction of the optimal reduction segment, the locality principles of each zone can also contribute to reducing the computational burden. The proposed algorithm eliminates points that do not satisfy the redefined locality principles of each zone. The orientation test, which determines whether a point is located to the left or right of a segment, is performed using the cross-product of vectors, requiring five subtractions and two multiplications. When employing the locality principles, the orientation test is performed exclusively on points located within the bounding box created by the two extremal points. In Figure 3b and Figure 3e, for instance, any points outside the bounding box formed by p_{l1} and p_{b1} are in violation of the locality principles. Hence, such points necessitate only a comparison of their coordinates without the need for an orientation test, which significantly reduces the overall number of computations.

C Estimation of Average-Case Time Complexity Using Harmonic Numbers

Estimating the average-case time complexity of the algorithm under investigation presents a significant challenge due to the unpredictable nature of its operations. The algorithm necessitates several traversals of the input data, with each traversal systematically reducing the number of points under consideration in the subsequent steps. The reduction rate of points is not uniform across iterations, but instead exhibits a decreasing trend. This irregularity in the reduction pattern introduces substantial complexity in deriving an accurate mathematical representation for the time complexity.

To address the challenge of estimating the average-case time complexity, we employ an approximation using harmonic numbers. Harmonic numbers, defined by the series $H(n) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$, mirror the number of operations for estimating the time complexity of our algorithm. We can consider $n * (\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n})$ as a rough estimate of the total operations, with each term approximating the number of remaining points in each iteration. The difference between two consecutive terms, $n * (\frac{1}{k} - \frac{1}{k+1})$, can be interpreted as the number of points eliminated in the k^{th} iteration. The approximation of the sum of the harmonic numbers is as follows [9]:

$$\sum_{k=1}^{n} \frac{1}{k} = \log n + \gamma + \frac{1}{2n} - \frac{1}{12n^2},$$
$$\text{where } \gamma \text{ is Euler's constant}$$

However, this approximation does not perfectly align with the algorithm's behavior. The algorithm often reduces more points than this model predicts, resulting in fewer remaining points than the harmonic series suggests. Consequently, the reduction step concludes before the n^{th} round, implying that the algorithm performs fewer operations than the sum of the harmonic numbers.

The harmonic series should be truncated at the point where the difference between the remaining points at two consecutive stages is less than one, to more accurately estimate the time complexity. This occurs at the $\sqrt{n}^{th}$ term [10], where the partial sum of the harmonic series up to this point is approximately half of the total sum. Hence, the number of operations for the reduction step up to the $\sqrt{n}^{th}$ term can be estimated as the sum of the operations for the following two processes: finding a reduction segment and performing the reduction.

$$\begin{aligned}
&(\frac{1}{1}n + \frac{1}{1}n) + (\frac{1}{2}n + \frac{1}{2}n) + (\frac{1}{3}n + \frac{1}{3}n) + \cdots + (\frac{1}{\sqrt{n}}n + \frac{1}{\sqrt{n}}n) \\
&= 2n(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\sqrt{n}}) \\
&\approx 2n(\log n + \gamma + \frac{1}{2n} - \frac{1}{12n^2}) \times \frac{1}{2} \\
&\approx n \times \log n
\end{aligned}$$

In conclusion, the average-case time complexity of the partition and reduction processes can be estimated as $O(n \log n)$ when there are n input points. This time complexity impacts the overall performance of the algorithm in average-case scenarios.

Analysis of Dynamic Voronoi Diagrams in the Hilbert Metric

Madeline Bumpus* Xufeng Caesar Dai† Auguste H. Gezalyan‡ Sam Muñoz§
Renita Santhoshkumar¶ Songyu Ye‖ David M. Mount**

Abstract

The objective of this paper is to study the properties of Hilbert bisectors and analyze Hilbert Voronoi diagrams in the dynamic setting. Additionally, we introduce dynamic visualization software for Voronoi diagrams in the Hilbert metric on user-specified convex polygons.

1 Introduction

In 1895, David Hilbert introduced the Hilbert metric [11], a projective metric used to measures distances within a convex body. It generalizes the Cayley-Klein model of hyperbolic geometry (on Euclidean balls) to any convex body. Hilbert geometry provides new insights into classical questions from convexity theory and the study of metric and differential geometries (such as Finsler geometries).

In particular, Hilbert Geometry is of interest in the field of convex approximation. To approximate convex bodies, many techniques [1–4, 8, 9, 14, 20] make use of covering convex bodies with regions that behave like metric balls in the Hilbert metric. These regions go by various names, such as: Macbeath regions, Macbeath ellipsoids, Dikin ellipsoids, and $(2, \varepsilon)$-covers. In this way, a deeper understanding of the Hilbert metric can lead us to a deeper understanding of convex approximation.

The Hilbert metric is applicable to many other fields of study such as quantum information theory [19], real analysis [13], optimization [6], and network analysis [7]. For example, the Hilbert metric is crucial in nonlinear Frobenius-Perron theory [7,12]. In addition, the Hilbert metric has often been used in the context of optimal mass transport and Schrödinger bridges [5–7, 18].

There has been little work on the design and analysis of algorithms in the Hilbert geometry on convex polygons and polytopes, and most of it has been done in the last two decades. In this paper, we build off of Nielson and Shao [15] as well as Gezalyan and Mount [10] to implement an algorithm for creating Voronoi diagrams in the Hilbert metric, characterize their bisectors as conics with an explicit formula, and analyze them in the dynamic setting.

*Howard Universty, Washington DC, USA, gmadeline.bumpus@bison.howard.edu
†Haverford College, Haverford, Pennsylvania, USA, xdai1@haverford.edu
‡Department of Computer Science, University of Maryland, College Park, USA octavo@umd.edu
§Colby College, Waterville, Maine, USA, smunoz23@colby.edu
¶Montgomery Blair High School, Silver Spring, Maryland, USA, renitadsanthoshkumar@gmail.com
‖Cornell Unversty, Ithaca, NY, USA, sy459@cornell.edu
**Department of Computer Science, University of Maryland, College Park, USA, mount@umd.edu

2 Preliminaries

2.1 Previous Work

In 2017, Nielsen and Shao [15] presented a characterization of the shape of balls in the Hilbert metric and explored their properties. They showed that the shape of a ball depends on the location its center as well as the vertices of the Hilbert domain. Nielsen and Shao gave an explicit description of Hilbert balls and studied the intersection of two Hilbert balls. In 2021, Gezalyan and Mount [10] expanded on this by studying the geometric and combinatorial properties of Voronoi diagrams in the Hilbert metric. They presented two algorithms, a randomized incremental and a divide and conquer algorithm, for constructing these diagrams. We extend these works to analyze other properties of Voronoi diagrams in the Hilbert geometry, and we develop a software package implementing an incremental algorithm based on the randomized incremental algorithm.

2.2 Defining the Hilbert Metric

A *convex body* $\Omega \subset \mathbb{R}^d$ is a closed, compact, full-dimensional convex set. Given two fixed points $s, t \in \mathbb{R}^d$, let $\|s-t\|$ denote the Euclidean distance and $H(s,t)$ denote the Hilbert distance between them. Let $\chi(s,t)$ denote the *chord* defined as the intersection of the line passing through s and t with Ω.

Definition 1 (Cross Ratio) *Given four distinct collinear points a, b, c, d in $\mathbb{R}^d$, their cross ratio is defined to be:*

$$(a, b; c, d) = \frac{\|a-c\|\|b-d\|}{\|b-c\|\|a-d\|}.$$

Where the orientation of the line determines the sign of each distance.

Note that the cross ratio is invariant under projective transformations [16].

Definition 2 (Hilbert metric) *Given a convex body Ω in $\mathbb{R}^d$ and two distinct points $s, t \in \operatorname{int}(\Omega)$, let x and y denote endpoints of the chord $\chi(s,t)$, so that the points are in the order $\langle x, s, t, y\rangle$. The Hilbert distance between s and t is defined as:*

$$H_\Omega(s,t) \;=\; \begin{cases} \frac{1}{2}\ln(s,t;y,x) & \text{if } s \neq t \\ 0 & \text{if } s = t. \end{cases}$$

It is well known that this is a metric, and in particular, it is symmetric and satisfies the triangle inequality. Moreover, it is well known that $H(a,b) + H(b,c) = H(a,c)$ if and only if $r_\Omega(a,b), r_\Omega(b,c), r_\Omega(a,c)$ are collinear and $r_\Omega(b,a), r_\Omega(c,b), r_\Omega(c,a)$ are collinear, where $r_\Omega(a,b)$ is the point where the ray ab meets $\partial(\Omega)$ [16, Theorem 12.4]. This follows from the Hilbert metric's characterization as being the average of the forward and backwards Funk metrics. It is also well known that geodesics are unique if and only if Ω is strictly convex [16, Corollary 12.7].

2.3 Voronoi Diagrams in the Hilbert Metric

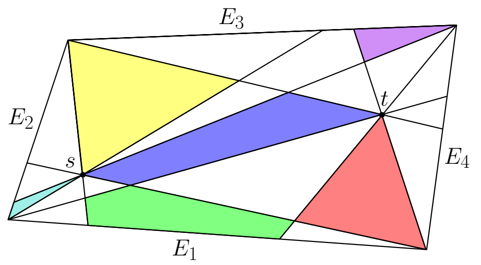
Let $\Omega \subset \mathbb{R}^2$ be an open convex region whose boundary, denoted $\partial\Omega$, is a polygon. Let S denote a set of n sites in Ω. The Voronoi cell of a site $s \in S$ is:

$$V(s) \;=\; \{q \in \Omega \,:\, d(q,s) \le d(q,t),\, \forall t \in S \setminus \{s\}\}.$$

The *Voronoi diagram* of S in the Hilbert metric induced by Ω, denoted $\mathrm{Vor}_\Omega(S)$, is the cell complex of Ω induced by the Voronoi cells $V(s)$, for all $s \in S$. Outside of the dynamic context, we assume that all sites are in general position, and in particular that no three sites are colinear. It is known that Voronoi cells in the Hilbert Metric are stars and with complexity $\Omega(mn)$ [10].

It will be helpful to define two terms, *spokes* and *sectors*. Given our convex body Ω and a site s in Ω it can be seen that the Hilbert distance from our site s to a point p is dependent on the edges of Ω that intersect $\chi(s,p)$. This is characterized by taking the vertices of Ω, which we denote by V, and partitioning Ω into cells determined by the chords $\chi(v,s)$ for all $v \in V$. We define a *spoke* $r_\Omega(v,s)$, $r_\Omega(s,v)$ to be the intersection of ray vs and sv respectively with Ω when v is a vertex of V. Nielsen and Shao showed that Hilbert balls are $\Theta(m)$-sided polygons whose boundaries are constructed around these *spokes* [15].

These *spokes* subdivide Ω into polygonal regions, which we will call *sectors*. Each sector T has the property that for all points $p \in T$, given two sites s and t there are four edges (not necessarily unique) of $\partial\Omega$, say E_A, E_B, E_C, E_D, such that $\chi(s,p)$ always intersects



🟦 $S(s,t,E_2,E_4,E_4,E_2)$ 🟥 $S(s,t,E_2,E_4,E_3,E_1)$
🟩 $S(s,t,E_2,E_1,E_4,E_1)$ 🟨 $S(s,t,E_1,E_3,E_4,E_2)$
🟪 $S(s,t,E_2,E_3,E_1,E_3)$ 🟦 $S(s,t,E_3,E_2,E_4,E_2)$

Figure 1: Illustration of different *sectors*.

Ω at E_A and E_B and $\chi(t,p)$ at E_C and E_D. We will write $S(s,t,E_A,E_B,E_C,E_D)$ to be the unique *sector* such that for any $p \in S(s,t,E_A,E_B,E_C,E_D)$, we have $\chi(sp)$ has in order of intersection E_A, s, p, E_B and $\chi(tp)$ has E_C, t, p, E_D (see Figure 1). As a consequence of this, we can see that the bisector between two sites, s and t, is composed of curves which depend piece-wise on the sectors they pass through.

3 Hilbert Bisector analysis

Theorem 1 *The equation of the bisector between two sites in any particular sector is of the form $Ax^2+Bxy+Cy^2+Dx+Ey+F=0$, with coefficients depending only on the line equations of the four edges and the sites.*

Proof. Suppose we are in some arbitrary sector $S(s,t,E_A,E_B,E_C,E_D)$. Let the equation of the edges be $a_1x+a_2y+a_3=0$, $b_1x+b_2y+b_3=0$, $c_1x+c_2y+c_3=0$, and $d_1x+d_2y+d_3=0$, for E_A, E_B, E_C, and E_D respectively.

Recall the Euclidean distance between a point (x,y) and a line $ux+vy+l=0$ is given by $D(u,v,l,x,y) = \frac{|ux+vy+l|}{\sqrt{u^2+v^2}}$. Let $p \in T$ be an arbitrary point with coordinates (p_x,p_y). Notice that, without loss of generality, if q is the point where chord $\chi(p,s)$ hits $\partial\Omega$ towards s, then the ratio $\frac{\|q-p\|}{\|q-s\|}$ can be replaced with the ratio $\frac{D(a_1,a_2,a_3,p_x,p_y)}{D(a_1,a_2,a_3,s_x,s_y)}$ by similar triangles. Using this technique we set $H(s,p) = H(t,p)$ and solve, yielding

$$\frac{D(a_1,a_2,a_3,p_x,p_y)}{D(a_1,a_2,a_3,s_x,s_y)}\frac{D(b_1,b_2,b_3,s_x,s_y)}{D(b_1,b_2,b_3,p_x,p_y)} = \frac{D(c_1,c_2,c_3,p_x,p_y)}{D(c_1,c_2,c_3,t_x,t_y)}\frac{D(d_1,d_2,d_3,t_x,t_y)}{D(d_1,d_2,d_3,p_x,p_y)}.$$

The equation of the line $ux+vy+l=0$ divides the plane in two, characterized by the sign of the function $g(x,y) = ux+vy+l$. Given this, since all our points are on the same side of each line, when we substitute in for the function D, we can remove the absolute value bars from our equations. It follows that:

$$\frac{a_1p_x + a_2p_y + a_3}{a_1s_x + a_2s_y + a_3}\frac{b_1s_x + b_2s_y + b_3}{b_1tp_x + b_2p_y + b_3} = \frac{c_1p_x + c_2p_y + c_3}{c_1t_x + c_2t_y + c_3}\frac{d_1t_x + d_2t_y + d_3}{d_1p_x + d_2p_y + d_3}.$$

Collecting constants on one side gives us:

$$\frac{b_1s_x + b_2s_y + b_3}{d_1t_x + d_2t_y + d_3}\frac{c_1t_x + c_2t_y + c_3}{a_1s_x + a_2s_y + a_3} = \frac{c_1p_x + c_2p_y + c_3}{a_1p_x + a_2p_y + a_3}\frac{b_1p_x + b_2p_y + b_3}{d_1p_x + d_2p_y + d_3}.$$

Letting the left side be a constant k, we find that the bisector satisfies the algebraic equation:

$$Ap_x^2 + Bp_xp_y + Cp_y^2 + Dp_x + Ep_y + F = 0,$$

where the coefficients are:

$$\begin{aligned}
A &= b_1c_1 - a_1d_1k \\
B &= b_2c_1 + b_1c_2 - a_1d_2k - a_2d_1k \\
C &= b_2c_2 - a_2d_2k \\
D &= b_3c_1 + c_3b_1 - a_3d_1k - a_1d_3k \\
E &= b_3c_2 + b_2c_3 - a_2d_3k - a_3d_2k \\
F &= b_3c_3 - a_3d_3k.
\end{aligned}$$

By the symmetry of these equations we can see that the bisector in $S(s, t, E_A, E_B, E_C, E_D)$ is the same as the bisector in $S(s, t, E_B, E_A, E_D, E_C)$.[1] □

We present several ways to simplify the bisector depending on the sector case.

3.1 Finding Bisectors in a Sector

There are three main sector types: sectors with four distinct edges, sectors with three, and sectors with two. We begin with the four edge case. It is a well known fact from projective geometry that there exists a projective transformation that maps any convex quadrilateral to another (see Figure 2). For the sake of completeness, we present the derivation of this transformation for the special case where the target is the unit square.

Observation 1 *Bisectors in the four-edge case can be projectively transformed and solved in the unit square.*

Proof. Suppose we are in a strictly four-edge sector $S(s, t, E_A, E_B, E_C, E_D)$ with edges E_A, E_C, E_B, E_D counter clockwise. Let our edges extend out infinity and let $p_1 = E_A \cap E_D$, $p_2 = E_A \cap E_C$, $p_3 = E_B \cap E_C$, and $p_4 = E_B \cap E_D$. Consider the vector:

$$Q^T = [q_{1,x}, q_{1,y}, q_{2,x}, q_{2,y}, q_{3,x}, q_{3,y}, q_{4,x}, q_{4,y}],$$

[1]We would like to thank Daniel Skora for comments on a previous version of this proof.

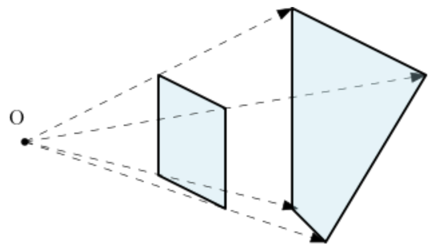


Figure 2: A projection between two quadrilaterals.

where $q_1 = (0, 0)$, $q_2 = (0, 1)$, $q_3 = (1, 1)$, and $q_4 = (1, 0)$. Then we can calculate the projective transformation matrix T such that $P' = TP$ when T sends points P on our quadrilateral to the unit square. We force the right corner of T to be 1 to fix our scaling factor.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We solve for the elements of T using the following matrices: $MV = Q$ where $V = [t_{11}, t_{12}, \ldots]^\intercal$, $Q = [q_{1,x}, q_{1,y}, \ldots]^\intercal$ and M is:

$$\begin{bmatrix}
p_{1,x} & p_{1,y} & 1 & 0 & 0 & 0 & -p_{1,x}\,q_{1,x} & -p_{1,y}\,q_{1,x} \\
0 & 0 & 0 & p_{1,x} & p_{1,y} & 1 & -p_{1,x}\,q_{1,y} & -p_{1,y}\,q_{1,y} \\
p_{2,x} & p_{2,y} & 1 & 0 & 0 & 0 & -p_{2,x}\,q_{2,x} & -p_{2,y}\,q_{2,x} \\
0 & 0 & 0 & p_{2,x} & p_{2,y} & 1 & -p_{2,x}\,q_{2,y} & -p_{2,y}\,q_{2,y} \\
p_{3,x} & p_{3,y} & 1 & 0 & 0 & 0 & -p_{3,x}\,q_{3,x} & -p_{3,y}\,q_{3,x} \\
0 & 0 & 0 & p_{3,x} & p_{3,y} & 1 & -p_{3,x}\,q_{3,y} & -p_{3,y}\,q_{3,y} \\
p_{4,x} & p_{4,y} & 1 & 0 & 0 & 0 & -p_{4,x}\,q_{4,x} & -p_{4,y}\,q_{4,x} \\
0 & 0 & 0 & p_{4,x} & p_{4,y} & 1 & -p_{4,x}\,q_{4,y} & -p_{4,y}\,q_{4,y}
\end{bmatrix}$$

Since the Hilbert metric is invariant under projective transformations, the final bisector can thus be computed for the canonical case of the unit square and then mapped back using the inverse transformation. □

Given this observation, we will assume henceforth that any four edges case is in the unit square.

Lemma 2 *When the four edges corresponding to a sector lie on four distinct lines, the bisector is either a hyperbola or a parabola.*

Proof. Let $S(s, t, E_A, E_B, E_C, E_D)$ be an arbitrary sector with four distinct edges. By applying Observation 1, we may assume without loss of generality that E_A, E_B, E_C, E_D lie on the lines $x = 0$, $x = 1$, $y = 0$, and $y = 1$, respectively. The discriminant of equation for our bisector is given by $B^2 - 4AC$. Substituting in for our variables, we get $(1 - k)^2$ as our discriminant, which is nonnegative. □

Next, let us consider the three-edge case. We will refer to the edge that is hit twice by the Hilbert metric in the sector as the *shared edge*, we have the following three cases, depending on whether the shared edge is behind both points with respect to the sector, in front of both (see Figure 3), or behind one and in front of the other (see Figure 4). As in the four-edge case, we can projectively transform the problem to a canonical form, as given in the following lemma. Define the *unit simplex* in $\mathbb{R}^2$ to be the right triangle with vertices at $(0,0)$, $(1,0)$, and $(0,1)$.

Observation 2 *Bisectors in the three-edge case can be projectively transformed and solved in the unit simplex.*

Proof. The affine transformation between triangles is well known and inherently projective. We provide it here. A triangle with corners p, q, r can be projectively transformed into the unit simplex with the matrix T^{-1}, where:

$$T = \begin{bmatrix} p_x - r_x & p_y - r_y & 0 \\ q_x - r_x & q_y - r_y & 0 \\ r_x & r_y & 1 \end{bmatrix}.$$

Observe that T^{-1} sends a point P on the left from our original triangle to the unit simplex. $\square$

Lemma 3 *Given any three-edge sector, where the shared edge is in front or behind of both sites, the bisector between the two sites in the sector is a straight line through the vanishing point of the non-shared edges.*

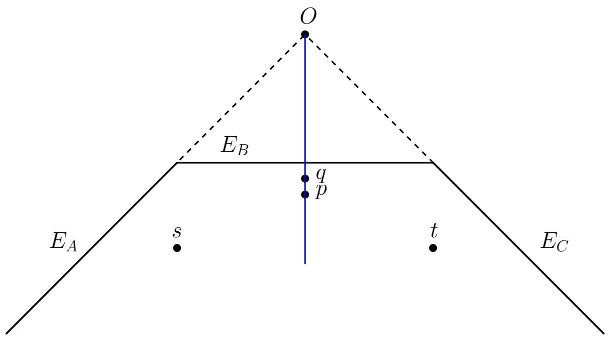


Figure 3: Illustration for the proof of Lemma 4 (a bisector in the three-edge case with shared edge in front of both sites).

Proof. We prove this geometrically for both cases $S(s,t,E_A,E_B,E_C,E_D)$ with edges $E_B = E_D$ and $S(s,t,E_A,E_B,E_C,E_D)$ with edges $E_A = E_C$.

Consider the case in which the edge is in front of both sites, $S(s,t,E_A,E_B,E_C,E_B)$. Let p be a point in the sector such that $H(s,p) = H(t,p)$, and let O be vanishing point of E_A and E_C. We claim the bisector is a straight line through Op. Let q be a point on the segment Op in the sector. Using the geodesic triangle inequality in the Funk metric we get $F(t,q) = F(t,p) + F(p,q)$ and $F(s,q) = F(s,p) + F(p,q)$ [16]. By definition, $F(s,p) + F(p,s) = F(t,p) + F(p,t)$. Substituting in our previous equations and cancelling $F(p,q)$ we have $F(s,q) + F(p,s) = F(t,q) + F(p,t)$. Using properties of similar triangles and rearranging the equation we can see that $F(s,q) - F(t,q) = \log((|pE_c|/|tE_c|) \cdot (|sE_A|/|pE_A|))$. Note that $|pE_c|/|pE_A| = |qE_c|/|qE_A|$. Substituting in and using the definition of the Funk metric we get $F(s,q) - F(t,q) = F(q,t) - F(q,s)$, yielding $H(s,q) = H(t,q)$. By choosing p to be the lowest point on the bisector in the sector we get our characterization of the bisector as a line.

Note the case where the shared edge is behind both sites follows from the symmetry of the general equation of the bisector. $\square$

Lemma 4 *Given a three-edge sector in which the shared edge is in front of one site and behind the other, the bisector between the sites can be a conic of any type (ellipse, parabola, or hyperbola) depending on their relative positions. Further, when either site is fixed, there exists a line such that the type of conic is determined by the location of the other site with respect to this line (an ellipse if it lies on one side, hyperbola if on the other side, and parabola if lies on the line).*

Proof. We prove this algebraically for, without loss of generality, $S(s,t,E_A,E_B,E_C,E_D)$ with edges $E_B = E_C$, and all other edges lying on distinct lines.

By Observation 2, it suffices to assume that the triangle has been mapped to the unit simplex. Let the edge E_A lie on the line $y = 0$, E_B and E_C lie on the line $x + y - 1 = 0$, and E_D lie on the line $x = 0$. By applying Theorem 1 with these values, the bisector is given by $x^2 + (2-k)xy + y^2 - 2x - 2y + 1 = 0$, where $k = \frac{s_x+s_y-1}{t_x}\frac{t_x+t_y-1}{s_y}$. The discriminant of the equation of the bisector is $(2-k)^2 - 4$, which reduces to $k(k-4)$. By standard results on conics, the type of conic depends on the sign of the discriminant. Note that k is always positive since (s_x, s_y) and (t_x, t_y) are in the unit simplex. Given this, the discriminant changes sign when $k = 4$. Thus, the conic type is given by the sign of the function $(s_x + s_y - 1)(t_x + t_y - 1) - 4s_x t_x$. Observe that when either s or t is fixed, this is a linear equation in coordinates of the other site. Thus, fixing s yields a line, and the type of conic (ellipse, hyperbola, or parabola) is determined by the location of t relative to this line. The elliptical case holds when t is on one side, the hyperbolic case when it is on the other, and the parabolic case when it lies on the line. This applies symmetrically for t. $\square$

Figure 4 illustrates four examples of conics as described by Lemma 4. The site coordinates and discriminant are given in Table 1. Note that the ellipse case and hyperbola case happen in general position.

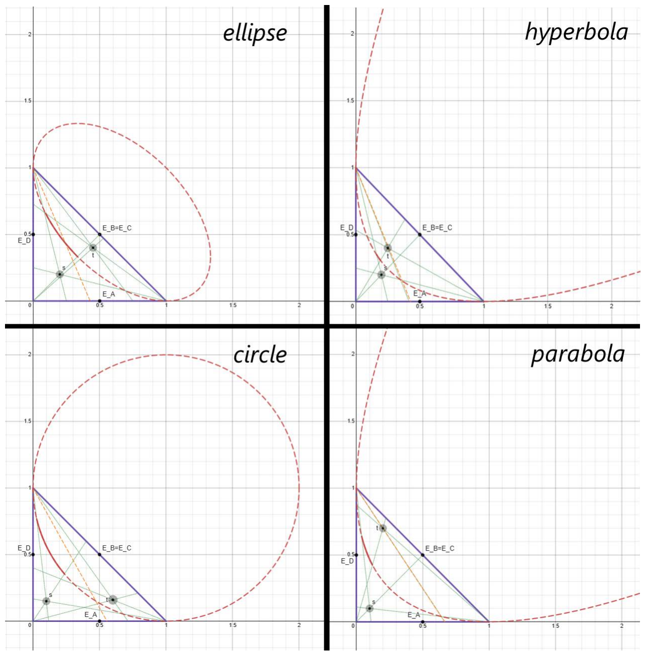


Figure 4: The bisector conics for the cases given in Table 1 visualized.

Table 1: The bisector cases illustrated in Figure 4 for the three-edge case.

	s	t	Discriminant
Ellipse	(0.2,0.2)	(0.45,0.4)	-3
Hyperbola	(0.2,0.2)	(0.25,0.4)	0.84
Circle	(0.1,0.15)	(0.6,0.16)	-4
Parabola	(0.1,0.1)	(0.2,0.7)	0

Now that we have evaluated the three-edge case we will evaluate the two-edge case.

Note that given a sector $S(s, t, E_A, E_B, E_C, E_D)$ with $E_A = E_D$ and $E_B = E_C$, we can choose a point p on E_A, q on E_B, and let $E_A \cap E_B = r$ and affinely map the triangle pqr into the unit simplex using Observation 2.

Lemma 5 *Given a two edge sector, $S(s, t, E_A, E_B, E_C, E_D)$ with $x = 0$ as $E_A = E_D$, and $y = 0$ as $E_B = E_C$, the bisector between s and t is given by the following linear equation: $y = \sqrt{k}x$ which passes through the vanishing point of E_A and E_B where $k = (s_y/t_x) \cdot (t_y/s_x)$.*

Proof. Plugging in to the general equation for the bisector in a sector give us $-kx^2 + y^2 = 0$ with $k = (s_y/t_x) \cdot (t_y/s_x)$ and solving gives us one viable solution $y = \sqrt{k}x$. It is clear that this goes through the vanishing point of E_A and E_B. □

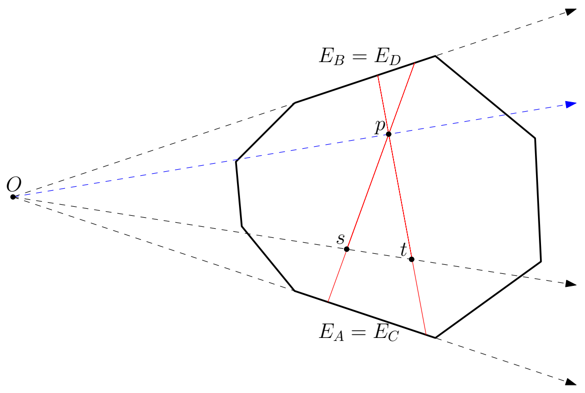


Figure 5: The point p is equidistant from both sites s and t by the invariance of the cross ratio.

4 Analysis of Hilbert Voronoi Diagrams

The Hilbert metric has some unique features that are not present in the Euclidean metric, In particular, in degenerate cases bisectors in the Hilbert metric can contain two dimensional polygons. Next, we present a necessary and sufficient condition for this occurrence.

Lemma 6 *The bisector between two sites s and t contains a Hilbert ball of some radius r if and only if both sites lie on a ray through a vertex V at the vanishing point of two edges E_A and E_B of Ω and the sector $S(s, t, E_A, E_B, E_B, E_A)$ exists.*

Proof. ($\Longleftarrow$) Consider that the Hilbert distance from s and t to any point p in $S(s, t, E_A, E_B, E_B, E_A)$, from the perspective of both s and t, is equal by the invariance of the cross ratio. Hence, the entire sector in which both s and t are between E_A and E_B is part of the bisector between s and t. By assumption, Ω is a bounded convex polygonal body, so our bisector will contain some fully 2-dimensional region. Note that we can always fit a Euclidean ball in any 2-dimensional region, and by we therefore must be able to fit a Hilbert ball in one as well [16].

($\Longrightarrow$) Suppose the bisector between two sites s and t contains a Hilbert ball N. Then there exists a ball N' nested in N so that N' lies entirely within one sector T. Note that the cross ratio between either site and any point in N' must be equivalent. We assert that T is defined by at most two edges of Ω.

Suppose that there were more than two unique edges defining T. Consider a point, p, in N'. Take a pair of edges with respect to one site and consider the ray from the vanishing point of that pair of edges through p. Note that moving p along this line will not change its distance to the site, however, since this sector is defined by more than two edges, it will from the other site. Hence, there can be only two edges of Ω defining T.

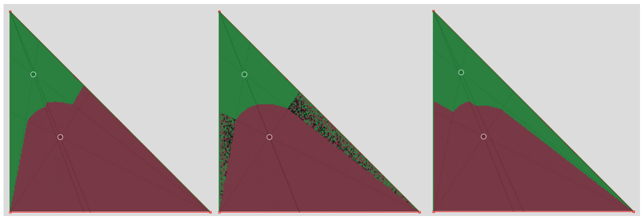


Figure 6: A discontinuity in the Hilbert metric Voronoi.

By the continuity of the Hilbert Metric, it must be that both sites lie on the same side of the bisector in T. Otherwise, we could move closer to one site and further from the other. Since both sites lie on the same side of their bisector, share two edges with respect to every point in N', and are the same distance to every point in N', they must lie on a ray through o. □

Given this, we can see that any dynamic Hilbert Voronoi diagram will contain discontinuities whenever a site passes over a ray from a vanishing point of a pair of edges and another site. (See Figure 6 for an example.)

Another property of Hilbert Voronoi diagrams that is distinct from Euclidean Voronoi diagrams is that three sites in general position do not necessarily create a Voronoi vertex. In particular given two site s and t, there exists a space of positive measure such no ball containing a point in that space on its boundary can contain both s and t on its boundary.

Lemma 7 *There exists a quadrilateral region Z of positive measure between any two sites s and t in a convex polygonal Ω, such that for all $r \in Z$, there is no Hilbert ball whose boundary passes through r, s and t.*

Proof. Let Ω have m edges, denoted $\{E_1, \ldots, E_m\}$. Let s and t be two sites in Ω. Take O to be the set of vanishing points on pairs of edges $O = \{O_{1,2}, O_{1,3}, \ldots, O_{m-1,m}\}$ when $O_{i,j}$ refers to the vanishing point of edges E_i and E_j. Let $R_{i,j}$ be the area between rays $O_{i,j}s$ and $O_{i,j}t$ intersected with Ω. Let $R = \{R_{1,2}, R_{1,3}, \ldots, R_{m-1,m}\}$.

Let $Z = \bigcap_R R_{i,j}$. By definition, Z is a polygon whose edges pass through both s and t, which always contains st. Note that it is equal to st if and only if s and t lie on a ray through an element of O.

Consider the boundary of Z. We begin by proving that Z is a quadrilateral. Without loss of generality let us consider only the top half of Z above the line st. Let the first two rays leaving s and t and forming part of the boundary of Z be denoted l_s and l_t, respectively. Let l_s and l_t intersect at a point v. If v is not in Z, there must exist some line l that intersects both l_s and l_t and either s or t, and therefore, must be one of the original rays l_s or l_t. This contradicts the fact that v is not in Z. Hence we know Z must be a quadrilateral. Next,

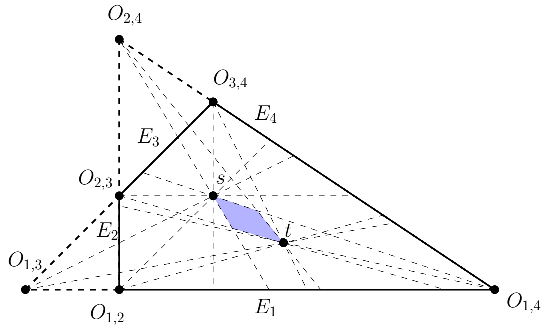


Figure 7: Illustration of region Z.

we prove that no ball through s and t can intersect the interior of Z on its boundary.

To do this, we prove that any ball containing s and t must contain Z. Consider an arbitrary Hilbert ball N with s and t on its boundary. Note that the boundary of N must be made of segments along rays passing through elements of O. Since s and t are on the boundary N, it must be that if an edge, W of N, intersects Z it would cut through, without loss of generality, the top half of the quadrilateral. Extending W, we see that it must intersect l_s and l_t at exactly one point each. Let W come from a ray passing through $O' \in O$. Consider segments through s and t created by the rays through O'. Note that W must be either above or below $O's$ and $O't$, or otherwise either s or t would not be on the boundary. However, Z is contained in the region between $O's$ and $O't$, so W cannot cut through it, contradiction. Hence we can see that N must contain the interior of Z. □

5 Software

In this section we present our software for Dynamic Voronoi Diagrams in the Hilbert metric.

Our software is built on the structure provided by the software presented by Nielsen and Shao [15]. We contributed by implementing a dynamic insertion algorithm. The insertion process works as follows.

Given an existing convex body Ω with a set of sites S, suppose the user chooses to insert a new site t. Our software will subdivide Ω into sectors from the perspective of t. Then for each site $s \in S$ our software calculates $\mathrm{Vor}_{\Omega,\{s,t\}}(t)$. This is done by tracing the bisector through the cell decomposition on Ω induced by the sectors of both s and t using the method described in [10]. While the bisector is approximated using line segments, the equation of the bisector is printed to the terminal window along with the sector information. Our algorithm will then update the Voronoi cell of all previously calculated $s \in S$'s as $\mathrm{Vor}_{\Omega,S\cup\{t\}}(s) = \mathrm{Vor}_{\Omega,S}(s) \cap \mathrm{Vor}_{\Omega,\{s,t\}}(s)$, and insert $\mathrm{Vor}_{\Omega,S}(t) = \bigcap_{s\in S} \mathrm{Vor}_{\Omega,\{s,t\}}(t)$ into the Voronoi diagram. Our software runs in quadratic time in terms of the number of sites and vertices of Ω and is available at https://github.com/caesardai/Voronoi_In_Hilbert/tree/main/src.

6 Concluding Remarks

In this paper, we presented an analysis of dynamic Voronoi diagrams in the Hilbert metric and presented software for the creation of such diagrams. As discussed in the introduction, the Hilbert metric is useful in the study of quantum information theory [19], real analysis [13], optimization [6], and network analysis [7]. Research in these fields may benefit from further extensions of algorithmic results in Euclidean to Hilbert geometry.

References

[1] Ahmed Abdelkader, Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Approximate nearest neighbor searching with non-euclidean and weighted distances. In *Proc. 30th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 355–372, 2019. doi:10.1137/1.9781611975482.23.

[2] Rahul Arya, Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Optimal bound on the combinatorial complexity of approximating polytopes. In *Proc. 31st Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 786–805, 2020. doi:10.1137/1.9781611975994.48.

[3] Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. Near-optimal ε-kernel construction and related problems. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 10:1–15, 2017.

[4] Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. On the combinatorial complexity of approximating polytopes. *Discrete Comput. Geom.*, 58(4):849–870, 2017. doi:10.1007/s00454-016-9856-5.

[5] Yongxin Chen, Tryphon Georgiou, and Michele Pavon. Optimal mass transport over bridges. In *Internat. Conf. Geom. Sci. of Inf.*, pages 77–84. Springer, 2015.

[6] Yongxin Chen, Tryphon Georgiou, and Michele Pavon. Entropic and displacement interpolation: a computational approach using the hilbert metric. *SIAM J. Appl. Math.*, 76(6):2375–2396, 2016.

[7] Yongxin Chen, Tryphon T Georgiou, Michele Pavon, and Allen Tannenbaum. Relaxed schrödinger bridges and robust network routing. *IEEE Trans. Contr. Netw. Sys.*, 7(2):923–931, 2019.

[8] Friedrich Eisenbrand, Nicolai Hähnle, and Martin Niemeier. Covering cubes and the closest vector problem. In *Proc. 27th Annu. Sympos. Comput. Geom.*, pages 417–423, 2011.

[9] Friedrich Eisenbrand and Moritz Venzin. Approximate CVPs in time $2^{0.802n}$. *Journal of Computer and System Sciences*, 2021.

[10] Auguste H. Gezalyan and David M. Mount. Voronoi diagrams in the Hilbert metric. In *Proc. 39th Internat. Sympos. Comput. Geom.*, pages 35:1–35:16, 2023. doi:10.4230/LIPIcs.SoCG.2023.35.

[11] D. Hilbert. Ueber die gerade linie als kürzeste verbindung zweier punkte. *Mathematische Annalen*, 46:91–96, 1895.

[12] Bas Lemmens and Roger Nussbaum. *Nonlinear Perron-Frobenius Theory*, volume 189. Cambridge University Press, 2012.

[13] Bas Lemmens and Roger Nussbaum. Birkhoff's version of hilbert's metric and its applications in analysis. *arXiv preprint arXiv:1304.7921*, 2013.

[14] Márton Naszódi and Moritz Venzin. Covering convex bodies and the closest vector problem. *arXiv preprint arXiv:1908.08384*, 2019.

[15] Frank Nielsen and Laetitia Shao. On balls in a Hilbert polygonal geometry (multimedia contribution). In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 67:1–67:4, 2017. doi:10.4230/LIPIcs.SoCG.2017.67.

[16] Athanase Papadopoulos and Marc Troyanov. From Funk to Hilbert geometry. *arXiv preprint arXiv:1406.6983*, 2014.

[17] Athanase Papadopoulos and Marc Troyanov. From Funk to Hilbert geometry. In *Handbook of Hilbert geometry*, volume 22 of *IRMA Lectures in Mathematics and Theoretical Physics*, pages 33–68. European Mathematical Society Publishing House, 2014. doi:10.4171/147-1/2.

[18] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[19] David Reeb, Michael J Kastoryano, and Michael M Wolf. Hilbert's projective metric in quantum information theory. *Journal of mathematical physics*, 52(8):082201, 2011.

[20] Thomas Rothvoss and Moritz Venzin. Approximate CVP in time $2^{0.802n}$ – now in any norm! *arXiv preprint arXiv:2110.02387*, 2021.

Every Combinatorial Polyhedron Can Unfold with Overlap

Joseph O'Rourke*

Abstract

Ghomi proved that every convex polyhedron could be stretched via an affine transformation so that it has an edge-unfolding to a net [Gho14]. A *net* is a simple planar polygon; in particular, it does not self-overlap. One can view his result as establishing that every combinatorial polyhedron $\mathcal{P}$ has a metric realization P that allows unfolding to a net.

Joseph Malkevitch asked if the reverse holds (in some sense of "reverse"): Is there a combinatorial polyhedron $\mathcal{P}$ such that, for every metric realization P in $\mathbb{R}^3$, and for every spanning cut-tree T of the 1-skeleton, P cut by T unfolds to a net? In this note we prove the answer is NO: every combinatorial polyhedron has a realization and a cut-tree that edge-unfolds the polyhedron with overlap.

1 Introduction

Joseph Malkevitch asked[1] whether there is a combinatorial type $\mathcal{P}$ of a convex polyhedron P in $\mathbb{R}^3$ whose every edge-unfolding results in a net. One could imagine, to use his example, that every realization of a combinatorial cube unfolds without overlap for each of its 384 spanning cut-trees [Tuf11].[2] The purpose of this note is to prove this is, alas, not true: every combinatorial type can be realized and edge-unfolded to overlap: Theorem 2 (Section 5). For an overlapping unfolding of a combinatorial cube, see ahead to Figure 12.

An implication of Theorem 2, together with [Gho14], is that a resolution of Dürer's Problem [O'R13] must focus on the geometry rather than the combinatorial structure of convex polyhedra.

2 Proof Outline

We describe the overall proof plan in the form of a multi-step algorithm. We will illustrate the steps with an icosahedron before providing details.

Algorithm. Realizing G to unfold with overlap.
Input: A 3-connected planar graph G.
Output: Polyhedron P realizing G and a cut-tree T that unfolds P with overlap.

(1) Select outer face B as base.
(2) Embed B as a convex polygon in the plane.
(3) Apply Tutte's theorem to calculate an equilibrium stress for G.
(4) Apply Maxwell-Cremona lifting stressed G to P.
(5) Identify special triangle $\triangle$.
(6) Compress P vertically to reduce curvatures (if necessary).
(7) Stretch P horizontally to sharpen the apex of $\triangle$ (if necessary).
(8) Form cut-tree T, including 'Z' around $\triangle$.
(9) Unfold $P \setminus T \to$ Overlap.

We are given a 3-connected planar graph G, which constitutes the combinatorial type of a convex polyhedron. By Steinitz's theorem, we know G is the 1-skeleton of a convex polyhedron. Initially assume G is triangulated; this assumption will be removed in Section 3.1.

(1) Select outer face B as base. Initially, any face suffices. Later we will coordinate the choice of B with the choice of the special triangle $\triangle$.

(2) Embed B as a convex polygon in the plane. Select coordinates for the vertices of B, which then pin B to the plane. B must be convex, but otherwise its shape is arbitrary.

(3) Apply Tutte's theorem [Tut63] to calculate an equilibrium stress—positive weights on each edge of G—that, when interpreted as forces, induce an equilibrium (sum to zero) at every vertex. This provides explicit coordinates for all vertices interior to B. The result is a Schlegel diagram, with all interior faces convex regions. Figure 1 illustrates this for the icosahedron.[3]

*Departments of Computer Science and of Mathematics, Smith College, Northampton, MA 01063, USA. `jorourke@smith.edu`.

[1]Personal communication, Dec. 2022.

[2]Burnside's Lemma can show that these 384 trees lead to 11 incongruent non-overlapping unfoldings of the cube [GSV19].

[3]Here the drawing is approximate, because I did not explicitly calculate the equilibrium stresses.

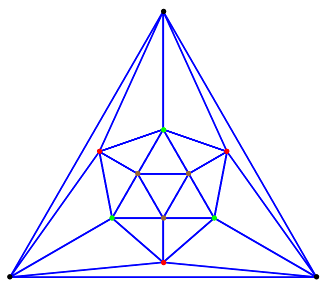


Figure 1: Icosahedron Schlegel diagram.

(4) Apply Maxwell-Cremona lifting to P. The Maxwell-Cremona theorem says that any straight-line planar drawing with an equilibrium stress has a polyhedral lifting via a "reciprocal diagram." The details are not needed here;[4] we only need the resulting lifted polyhedron. An example from [Sch08] shows the vertical lifting of a Schlegel diagram of the dodecahedron: Figure 2. A lifting of the vertices of the icosahedron in Figure 1 is shown in Figure 3.[5]

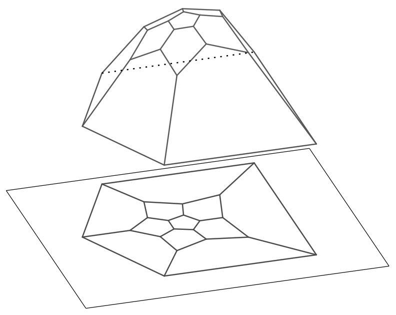


Figure 2: Maxwell-Cremona lifting to a dodecahedral diagram. [Sch08], by permission of author.

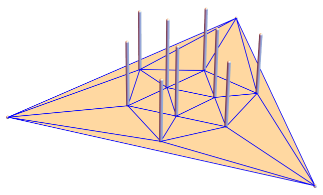


Figure 3: Vertical lifting the vertices of the icosahedron Schlegel diagram in Figure 1.

(5) Identify special triangle $\triangle$. This special triangle must satisfy several conditions, which we detail later (Section 3). For now, we select $\triangle = a_1a_2a_3 = 6, 8, 5$ in Figure 4.

[4]A good resource on this topic is [RG06].

[5]This is again an approximation as I did not calculate the reciprocal diagram.

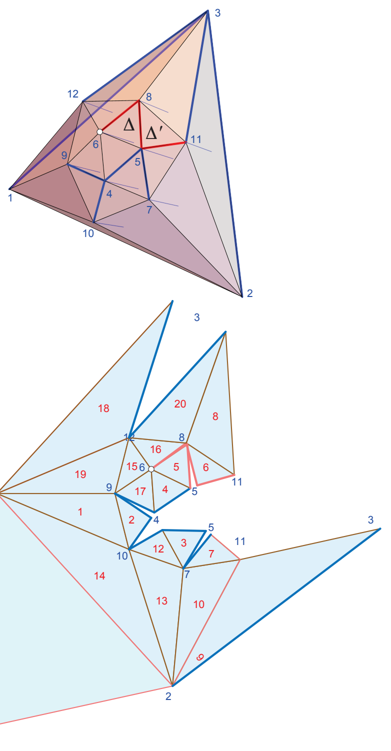


Figure 4: Red: face numbers; blue: vertex indices. $\triangle = 5$, $\triangle' = 6$. Z-portion of spanning tree T red; remainder blue.

(6) Compress P vertically (if necessary) to reduce curvatures. Not needed in icosahedron example.

(7) Stretch P horizontally (if necessary) to sharpen apex of $\triangle$. Not needed in icosahedron example.

(8) Form cut-tree T, including a 'Z'-path around $\triangle$. We think of a_1 as the root of the spanning tree, which includes the Z-shaped (red) path $a_1a_2a_3a_4$ around $\triangle$ and the adjacent triangle $\triangle'$ sharing edge a_2a_3. In Figure 4, the Z vertex indices are $6, 8, 5, 11$. The remainder of T is completed arbitrarily.

(9) Unfold $P \setminus T$. Finally, the conditions on $\triangle$ ensure that cutting T unfolds P with overlap along the a_2a_3 edge. See Figure 5.

3 Conditions on $\triangle$

We continue to focus on triangulated polyhedra. In order to guarantee overlap, the special triangle $\triangle = a_1a_2a_3$ should satisfy several conditions:

1. The angle at a_2 in $\triangle$ must be $\leq \pi/3 = 60°$, and the edge a_2a_3 at least as long as a_1a_2.

2. The spanning cut-tree T must contain the Z as previously explained. In addition, no other edge of T

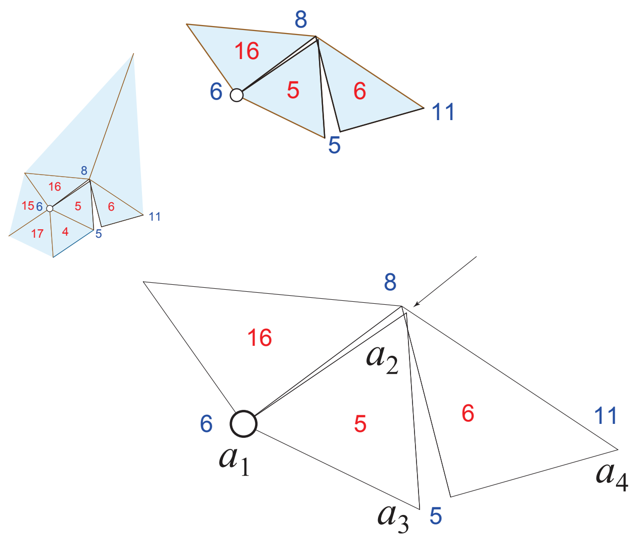


Figure 5: Close-up views of overlap.

is incident to either a_1 or a_2. In particular, edge a_1a_3 is not cut, so the triangle $\triangle$ rotates as a unit about a_1.

3. The curvatures at a_1 and a_2 must be small. (The *curvature* or "angle gap" at a vertex is 2π minus the sum of the incident face angles.) We show below that $< 20°$ suffices.

4. $\triangle$ should be disjoint from the base B: $\triangle$ and B share no vertices.

This 4th condition might be impossible to satisfy, in which case an additional argument is needed (Section 4). For now we concentrate on the first three conditions.

$\triangle$ is chosen to be the triangle disjoint from B with the smallest angle α. Clearly $\alpha \le \pi/3 = 60°$. Let $\triangle = a_1a_2a_3$ with a_2 the smallest angle. Chose the labels so that $|a_1a_2| \le |a_2a_3|$. It will be easy to see that $\triangle$ an equilateral triangle is the "worst case" in that smaller α lead to deeper overlap, and $|a_1a_2| = |a_2a_3|$ suffices for overlap. So we will assume $\triangle$ is an equilateral triangle.

Next, we address the requirement for small curvatures, when the second condition is satisfied: no other edge of T is incident to either a_1 or a_2. Let ω be the curvature at both a_1 and a_2. Then an elementary calculation shows that $\omega = \frac{1}{9}\pi = 20°$ would just barely avoid overlap: see Figure 6.

One can view the flattening of a_1 and a_2 when cut as first turning the edge a_2a_3 by ω about a_2, and then rotating the rigid path $a_1a_2a_3'$ about a_1 by ω. For any ω strictly less than 20°, overlap occurs along the a_2a_3 edge: Figure 6(b). The basic reason this "works" to create overlap is that the cut-path around $\triangle$ is not *radially monotone*, a concept introduced in [O'R16] and used in [O'R18] and [Rad21] to avoid overlap.

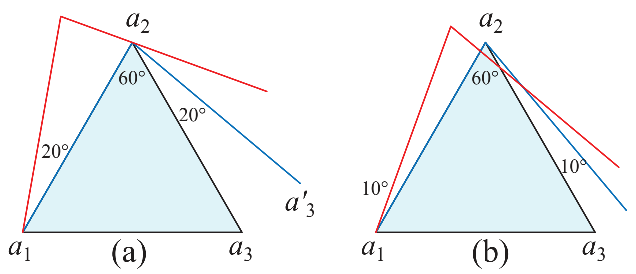


Figure 6: (a) $\omega = 20°$ avoids overlap. (b) $\omega = 10°$ overlaps.

In the unfolded icosahedron in Figure 4, the angle at a_2 is 59°, and the curvatures ω_1, ω_2 at a_1, a_2 are 2.4° and 8.1° respectively.

If the two curvatures are not less than 20°, then we scale P vertically, orthogonal to base B, step (6) of Algorithm 2. As illustrated in Figure 7, this flattens dihedral angles and reduces vertex curvatures (which reflect the spread of the normals [Hor84]) at all but the vertices of base B, which increase to compensate the Gsuss-Bonnet sum of 4π. Clearly we can reduce curvatures as much as desired.

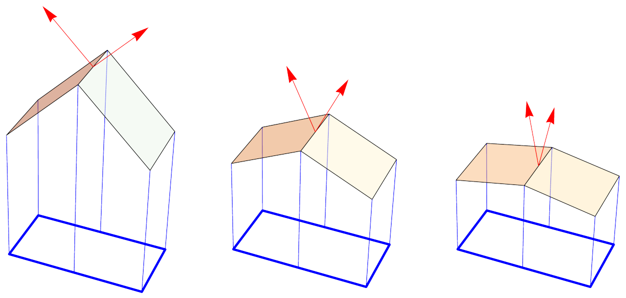


Figure 7: Dihedral angle δ flattens as z-heights scaled: $(1, \frac{1}{2}, \frac{1}{5}) \rightarrow (90°, 125°, 160°)$.

3.1 Non-Triangulated Polyhedra

If G and therefore P contains non-triangular faces, then we employ step (7) of Algorithm 1: Scale P horizontally, parallel to the xy plane containing B. For example, in the dodecahedron example (Figure 2), no face has an angle $\alpha \le \pi/3$. The following lemma shows we can sharpen any selected face angle.

Lemma 1. *Any face angle $\angle a_1a_2a_3$ can be reduced via an affine stretching transformation to be arbitrarily small.*

Proof. Adjust the coordinate system so that a_1a_3 lies in the yz-plane containing the origin, with a_2 in the x-positive halfspace, wlog at $a_2 = (1, a_{2y}, a_{2z})$. See Figure 8. The Tutte-embedding guarantees that $\triangle a_1a_2a_3$ is not degenerate—the three vertices are not collinear, and Maxwell-Corona lifting guarantees the triangle is not vertical because each vertex of the Schlegel diagram lies in the relative interior of its neighbors [RG06, p.126,136]. Now stretch all vertices by $s > 1$ in their x-coordinate. This leaves a_1 and a_3 fixed, while a_2 stretches horizontally to $a'_2 = (s, a_{2y}, a_{2z})$. Eventually with large s the angle $\angle a_1a'_2a_3$ decreases monotonically to zero, while maintaining $|a_1a'_2| \leq |a'_2a_3|$. $\square$

So we can identify a $\triangle$ within any face, stretch its angle below 60°, and proceed just as in a triangulated polyhedron: Because a_1a_3 is not cut, having $\triangle$ joined to a triangle below (4 in Figure 4) is no different than having $\triangle$ part of a face.

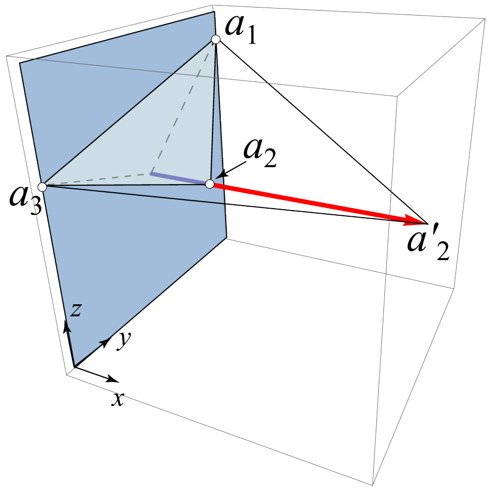


Figure 8: Stretching $\angle a_1a_2a_3 = 108°$ to $\angle a_1a'_2a_3 = 53°$.

4 No Pair of Disjoint Faces

Finally we focus on the 4th condition that $\triangle$ should be disjoint from the base B. If G contains any two disjoint faces, triangles or k-gon faces with $k > 3$, we select one as B and the other to yield $\triangle$. So what remains is those G with no pair of disjoint faces.

For example, a pyramid—a base convex polygon plus one vertex a (the apex) above the base—has no pair of disjoint faces. However, note that a pyramid has pairs of faces that share one vertex but not two vertices. It turns out that this suffices to achieve the same structure of overlap. Figure 9 illustrates why. Here B is a triangle $b_1b_2a_3$ and we select $\triangle = a_1a_2a_3$. The small-curvature requirement holds just for a_1, a_2—the start of the Z—the curvature at a_3 could be large (117° in this example) but does not play a role, as the unfolding illustrates. Therefore, if G has no pair of disjoint faces, but does have a pair of faces that share a single vertex, we proceed just in Algorithm 1, suitably modified.

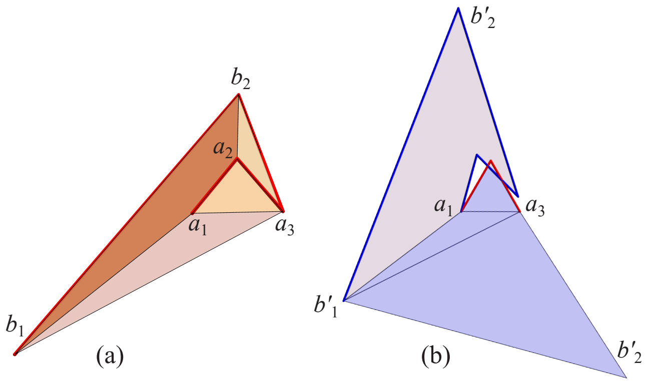


Figure 9: (a) B and $\triangle$ share a_3. Z $= a_1a_2a_3b_2$. (b) Unfolding with overlap.

For the pyramid example, two triangles sharing just the apex would serve as $\triangle$ and base B. Consider the square pyramid in Figure 10(a), with B and $\triangle$ marked. Mapping $\triangle = 145$ to $\triangle = a_1a_2a_3$ at the shared pyramid apex, (b) of the figure shows that this is equivalent to Figure 9(a). A hexagonal pyramid is illustrated in the Appendix.

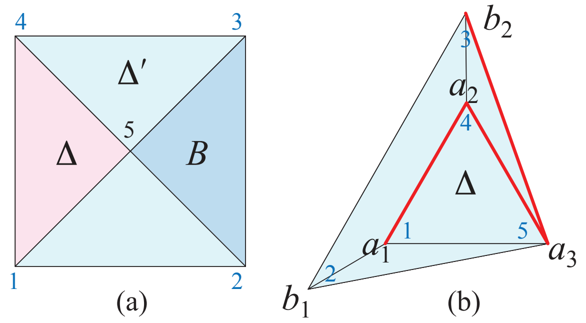


Figure 10: (a) Square pyramid Schlegel diagram, apex 5, square base 1234. (b) Relablled to match Figure 9(a).

This leaves the case where there are no two disjoint faces, nor two faces that share just a single vertex: every pair of faces shares two or more vertices. If two faces share non-adjacent vertices, they cannot both be convex. So in fact the condition is that each two faces share an edge. Then, it is not difficult to see that G can only be a tetrahedron, as the following argument shows.

Start with Euler's formula, $V - E + F = 2$. Each vertex v must be incident to exactly three faces, because, if v has degree ≥ 4, then each non-adjacent pair of faces incident to v cannot share an edge. So $3V = 2E$. Substituting into Euler's formula yields $F = 2 + E/3$.

Because each pair of faces share an edge, $F(F-1)$ double counts edges:[6] $2E = F(F-1)$. Substituting,

$$\begin{aligned} F &= 2 + E/3 \\ E &= F(F-1)/2 \\ F &= 2 + F(F-1)/6 \\ F^2 - 7F + 12 &= 0 \end{aligned}$$

The two solutions of this quadratic equation are $F = 3$, which cannot form a closed polyhedron, and $F = 4$. The tetrahedron is the only polyhedron with four faces, and indeed $F = 4$ implies $V = 4$ and $E = 6$.

So the only case remaining is a tetrahedron. But it is well known that the thin, nearly flat tetrahedron unfolds with overlap: Figure 11. And since there is only one tetrahedron combinatorial type, this completes the inventory.

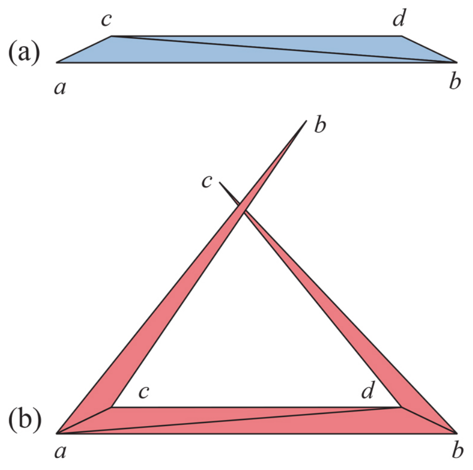


Figure 11: Figure 28.2 [detail], p.314 in [DO07]: tetrahedron overlap. Blue: exterior. Red: interior. Cut tree $T = abcd$. (T is a combinatorial 'Z'.)

5 Theorem

We have proved this theorem:

Theorem 2. *Any 3-connected planar graph G can be realized as a convex polyhedron P in $\mathbb{R}^3$ that has a spanning cut-tree T such that the edge-unfolding of $P \setminus T$ overlaps in the plane.*

So together with Ghomi's result,[7] any combinatorial polyhedron type can be realized to unfold and avoid overlap, or realized to unfold with overlap.

[6]Similar logic is used to form Szilassi's polyhedral torus.
[7]See [SZ18] for a different proof of [Gho14].

Returning to Malkevitch's example of a combinatorial cube, consider Figure 12. Starting from the standard Schlegel diagram for a cube (one square inside another (B), trapezoid faces between the squares), horizontal stretching (step (7) of Algorithm 1) is applied to squeeze the top and bottom squares to 1×2 and 2×4 diamonds, so that the angle at a_2 becomes small, in this case $2 \arctan(1/2) \approx 53°$. The lifting leaves the curvatures at a_1, a_2 to be small enough, $6.0°, 6.5°$, so the vertical scaling step (6) of Algorithm 1 is not needed.

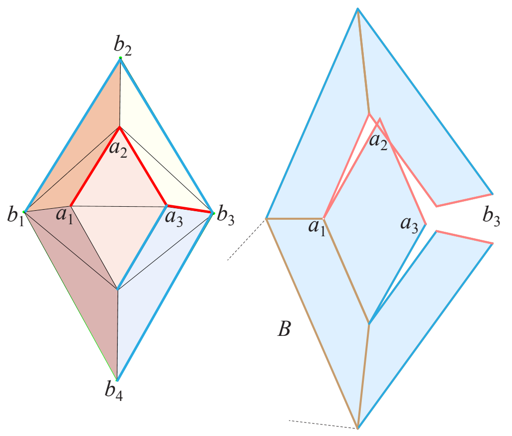


Figure 12: Unfolding of a combinatorial cube. Diagonals in the left figure are an artifact of the software; all lateral faces are planar congruent trapezoids. Base B attached left of b_1b_4 not shown. Vertex coordinates:

$$(-1,0,0.5), (1,0,0.5), (0,-2,0.5), (0,2,0.5),$$
$$(-2,0,0), (2,0,0), (0,-4,0), (0,4,0)$$

6 Open Problem

Is there a combinatorial type $\mathcal{P}$ of a Hamiltonian polyhedron (i.e., one with a Hamiltonian path), such that, for every metric realization $P \subset \mathbb{R}^3$, and every Hamiltonian path T, $P \setminus T$ unfolds to a net?

This restricts Malkevitch's question to combinatorial Hamiltonian polyhedra $\mathcal{P}$, and restricts T to a Hamiltonian path, producing a *zipper unfolding* [DDL+10]. Note: Some convex polyhedra are not Hamiltonian, e.g., the rhombic dodecahedron.

To rephrase the question: Is there a combinatorial Hamiltonian polyhedron whose every metric realization and zipper unfolding avoids overlap? Or is there instead an analog of Theorem 2 showing that even under these restrictions, there is always a realization and a zipper unfolding that overlaps?

Acknowledgements. I benefitted from discussions with Richard Mabry and Joseph Malkevitch, and three referees. In particular, one referee suggested Lemma 1 to repair an oversight.

References

[DDL+10] Erik D. Demaine, Martin L. Demaine, Anna Lubiw, Arlo Shallit, and Jonah Shallit. Zipper unfoldings of polyhedral complexes. In *Proc. 22nd Canad. Conf. Comput. Geom.*, pages 219–222, August 2010.

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

[Gho14] Mohammad Ghomi. Affine unfoldings of convex polyhedra. *Geometry & Topology*, 18(5):3055–3090, 2014.

[GSV19] Richard Goldstone and Robert Suzzi Valli. Unfoldings of the cube. *The College Mathematics Journal*, 50(3):173–184, 2019.

[Hor84] Berthold K. P. Horn. Extended Gaussian images. *Proceedings of the IEEE*, 72(12):1671–1686, 1984.

[O'R13] Joseph O'Rourke. Dürer's problem. In Marjorie Senechal, editor, *Shaping Space: Exploring Polyhedra in Nature, Art, and the Geometrical Imagination*, pages 77–86. Springer, 2013.

[O'R16] Joseph O'Rourke. Unfolding convex polyhedra via radially monotone cut trees. *arXiv:1607.07421*, 2016. `https://arxiv.org/abs/1607.07421`.

[O'R18] Joseph O'Rourke. Edge-unfolding nearly flat convex caps. In *Proc. Symp. Comput. Geom. (SoCG)*, volume 99, pages 64:1–64:14. Leibniz Internat. Proc. Informatics, June 2018. Full version: `https://arxiv.org/abs/1707.01006`.

[Rad21] Manuel Radons. Edge-unfolding nested prismatoids. *arXiv:2105.00555*, 2021.

[RG06] Jürgen Richter-Gebert. *Realization Spaces of Polytopes.* Springer, 2006.

[Sch08] André Schulz. *Lifting planar graphs to realize integral 3-polytopes and topics in pseudo-triangulations.* PhD thesis, Univeritât Berlin, 2008.

[SZ18] Gözde Sert and Sergio Zamora. On unfoldings of stretched polyhedra. *arXiv:1803.09828*, 2018. `https://arxiv.org/abs/1803.09828`.

[Tuf11] Christopher Tuffley. Counting the spanning trees of the 3-cube using edge slides. *arXiv:1109.6393*, 2011. `https://arxiv.org/abs/1109.6393`.

[Tut63] William T. Tutte. How to draw a graph. *Proc. London Mathematical Society*, 13(52):743–768, 1963.

A Hexagonal Prism

Figure 13 shows a hexagonal prism, following the model of the square prism in Figure 10: no pair of faces are disjoint, but $\triangle$ and B marked share just one vertex. Figure 14 shows its overlapping unfolding.

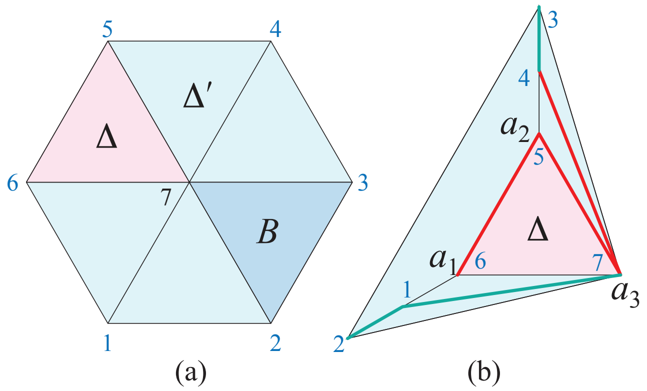


Figure 13: (a) Schlegel diagram of a hexagonal prism. (b) Overhead view of combinatorial rearrangement. The cut tree T is marked with red and blue paths.

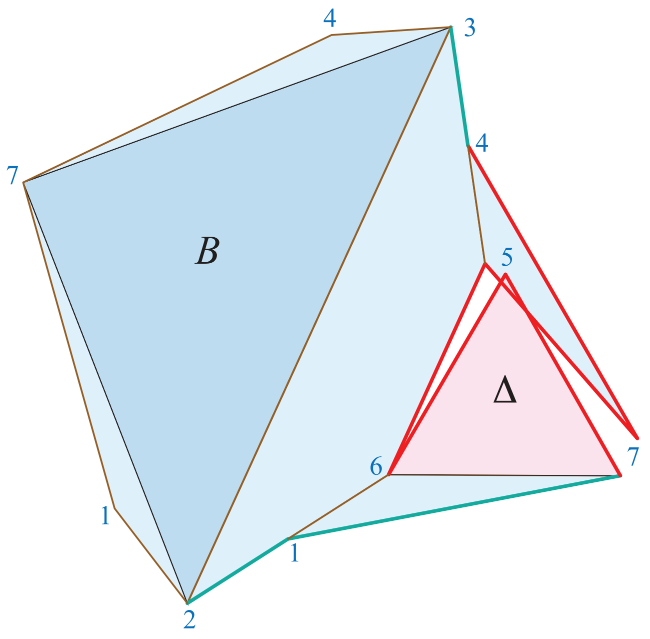


Figure 14: Unfolding of Figure 13(b). Curvature at v_6 and v_5 is 5.7°. Vertices 1 and 4 are collinear with 26 and 35 respectively. Hexagon: 123456.

Reconfiguration of Linear Surface Chemical Reaction Networks with Bounded State Change

Robert M. Alaniz* Michael Coulombe† Erik D. Demaine† Bin Fu* Timothy Gomez†
Elise Grizzell* Ryan Knobel* Andrew Rodriguez* Robert Schweller* Tim Wylie*

Abstract

We present results on the complexity of reconfiguration of surface Chemical Reaction Networks (sCRNs) in a model where surface vertices can change state a bounded number of times based on a given burnout parameter k. We primarily focus on linear $1 \times n$ surfaces. Without a burnout bound, or even with an exponentially high bound on burnout, reconfiguration on linear surfaces is known to be PSPACE-complete. In contrast, we show that the problem becomes NP-complete when the burnout k is polynomially bounded in n. For smaller $k = O(1)$, we show the problem is polynomial-time solvable, and in the special case of $k = 1$ burnout, reconfiguration can be solved in linear $O(n + |R|)$ time, where $|R|$ denotes the number of system rules. We additionally explore some extensions of this problem to more general graphs, including a fixed-parameter tractable algorithm in the height m of an $m \times n$ rectangle in 1-burnout, a polynomial-time solution for 1-burnout in general graphs if reactions are non-catalytic, and an NP-complete result for 1-burnout in general graphs.

1 Introduction

A prominent area of research in molecular computation, Chemical Reaction Networks (CRNs), study well-mixed solutions of molecules. Limited by the inherent lack of geometry, the model has important restrictions on its computational power, including no proven capability of error-free computation of logarithm [6] or Turing universality [16]. Specifically, CRNs are capable of computing all semilinear functions [5]. The introduction of a surface and, by extension, geometry, with abstract Surface Chemical Reaction Networks (sCRNs) removes these limitations, and thus has increased computational power. Molecular computing on a surface is an increasingly popular direction in both experimental [4, 18] and theoretical [10, 13] research.

In this paper, we explore a restricted version of the powerful surface CRN model, where each molecule in the system can only change in a reaction a set number of times. We refer to this constraint as *burnout.* Bounding the number of state changes leads to polynomial-time and XP algorithms for many reconfiguration problems that are otherwise PSPACE-complete.

Motivations for the study of problems with burnout include examples such as optimizing limited lifetime biomolecules or modeling redox reactions in which the electron transfer from one chemical species to another increases the cost of further reaction beyond what any other current or future neighbors could afford.

1.1 Previous Work

Surface Chemical Reaction Networks (sCRNs) were introduced in [15] with a simulator provided in [7]. These papers show various constructions such as Boolean circuits and a Cellular Automata simulation.

Another restricted version of sCRNs uses only swap reactions, in which the two species only change position, Example: $A + B \rightarrow B + A$. In [2], the authors show swap reactions are capable of feed-forward computation and provide an analysis of thermodynamic properties of the circuit. Recently, [1] showed that reconfiguration is PSPACE-complete for swap surface CRNs with only 4 species and 3 reactions, and in P with any system of fewer species or reactions. This work also introduces k-burnout surface CRNs and show two important results: that 1-reconfiguration (whether a single cell can change) is NP-complete with 1-burnout and that general reconfiguration is NP-complete with 2-burnout. Burnout is similar to the freezing concept from Cellular Automata [11, 12, 17] and Tile Automata [3], but while freezing is defined as having an ordering on states or a tile never revisiting a state, burnout is a constraint where a cell never reacts more than a fixed number of times. Thus, returning to a previous state is possible, unlike the freezing restrictions.

1D Cellular Automata are capable of Turing computation from [8]. P-completeness of prediction, is this cell in state at time step less than t, for Cellular Automata Rule 110 was shown in [14], implying it is also capable of efficient computation. This problem is also P-complete for a number of Freezing CAs in 2D, while it is always in NL for Freezing 1D CAs [12]. This work also gives a 1D freezing CA, which is Turing universal.

*Department of Computer Science, University of Texas Rio Grande Valley

†Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology

Shape	Burnout	Result	Theorem
$1 \times n$	1	$O(n + \|R\|)$	Thm. 1
$1 \times n$	2	$O(n \cdot \|S\|^2 \cdot \|R\|^4)$	Thm. 2
$1 \times n$	$O(1)$	P for $O(1)$ degree	Thm. 3
$1 \times n$	k (unary)	NP-complete	Thm. 4
$1 \times n$	Unbounded	PSPACE-complete	[15]
Planar	1	$O(\|V\|^{1.5} + \|R\|)$	Thm. 5**
General	1	NP-complete	Thm. 7
$m \times n$	1	NP-complete	[1]‡
$m \times n$	1	FPT in m	Thm. 8‡

Table 1: Comparison of reconfiguration results. For a CRN system, R is the set of rules and S is the set of species. V is the set of vertices for the graph defining the shape. **Non-catalytic rules only. ‡These results are for the problem of 1-Reconfiguration.

1.2 Our Contributions

This work investigates the reconfiguration problem for linear surface CRNs with k-burnout. Our results are outlined in Table 1. We begin in Section 3, where we present a polynomial-time algorithm for 1D 1-burnout. We then increase the burnout number to investigate 1D 2-burnout systems and prove that this is still in P. Following this, we show that for the case of any fixed $k = \mathcal{O}(1)$, there exists an algorithm that has a polynomial runtime. In the terms of parameterized complexity classes, this is the class XP, also known as slice-wise polynomial [9]. We then present an NP-completeness proof for when the burnout is a unary input. This result contrasts PSPACE-completeness known when the burnout is unbounded or exponentially high [15].

After $1 \times n$ lines, we begin investigating 1-burnout in 2D systems in Section 5. We start with the problem of reconfiguration, where we only have non-catalytic rules. We then show that on an arbitrary graph and with all types of rules, the reconfiguration problem is NP-complete. Finally, we study the problem of 1-reconfiguration for bounded-height surfaces, presenting an XP algorithm parameterized by height.

2 Preliminaries

A brief overview of the model and relevant problems.

Surface, Cells and Species. A *surface* for a CRN Γ is an undirected graph G of large size n. The vertices of the surface are also referred to as *cells*. Many of our results deal with $1 \times n$ grid graphs, or *linear* surfaces.

The state of a vertex is representative of a molecular *species* in the system. Chemical *reactions* considered here are bimolecular, as in they occur between two species in neighboring vertices. A rule denoting that neighboring species A and B may react to become C and D is written as $A + B \to C + D$. This is a **non-catalytic** rule, as both species change. In a **catalytic** reaction, only one of the two species will change, e.g.

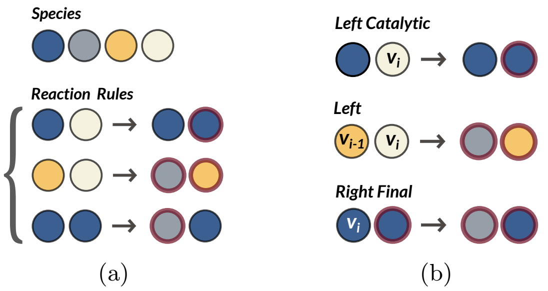


Figure 1: (a) An example sCRN system with 4 species, three rules, and 1 burnout. (b) Rule types used in Figure 2 example. *Note: The red ring outline shows whether the vertex has been "burned out." There is no effect on the reaction rule itself.*

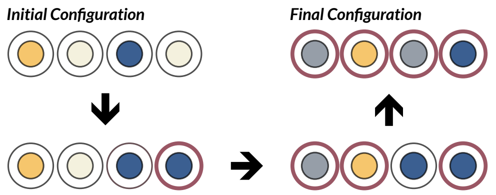


Figure 2: A possible sequence of reactions for the system described in Figure 1

$C + D \to C + B$, the other used as a catalyst.

A **surface Chemical Reaction Network (sCRN)** consists of a surface, a set of molecular species S, and a set of reaction rules R. A *configuration* is a mapping from each vertex to a species from the set S.

Reachable Configurations. For two configurations I, T, we write $I \to^1_\Gamma T$ if there exists a $r \in R$ such that performing reaction r on a pair of species in I yields the configuration T. Let $I \to_\Gamma T$ be the transitive closure of $I \to^1_\Gamma T$, including loops from each configuration to itself. Let $\Pi(\Gamma, I)$ be the set of all configurations T where $I \to_\Gamma T$ is true.

Burnout. A limit on the number of changes that can occur in any vertex v_i. In systems that allow catalytic reactions, after this limit has been reached, while v_i will not change again, neighboring species may still use the species in that cell as a catalyst.

Reconfiguration Problem. Given an sCRN Γ and two configurations I and T, is $T \in \Pi(\Gamma, I)$?

1-Reconfiguration Problem. Given an sCRN Γ, configuration I, vertex v, and species s, does there exist a $T \in \Pi(\Gamma, I)$ such that T has species s at vertex v?

3 Algorithms for Constant Burnout

We show that reconfiguration of a linear surface is solvable in polynomial-time when the burnout is one or two.

3.1 1-Burnout Linear Surfaces

In the case of 1-burnout with a $1 \times n$ line, the problem of reconfiguration is solvable in linear time with respect to n and the size of the rule set. As an observation, there are at most six reactions for any vertex, v_i, on a linear surface since a vertex has at most two neighbors. These reactions include the following:

- A left reaction, where vertex v_i reacts with vertex v_{i-1} and both vertices reach their final states.
- A left catalytic reaction, where vertex v_i reacts with vertex v_{i-1} in its initial state to transition vertex v_i to its final state without changing v_{i-1}.
- A left final-catalytic reaction (or left final), where vertex v_i reacts with vertex v_{i-1} in its final state to transition vertex v_i to its final state without changing v_{i-1}.
- A right reaction, where vertex v_i reacts with vertex v_{i+1} and both vertices reach their final states.
- A right catalytic reaction, where vertex v_i reacts with vertex v_{i+1} in its initial state to transition vertex v_i to its final state without changing v_{i+1}.
- A right final-catalytic reaction (or right final), where vertex v_i reacts with vertex v_{i+1} in its final state to transition vertex v_i to its final state without changing v_{i+1}.

Additionally, we also consider when a vertex is in its final state. An example system and sequence of reactions can be found in Figures 1 and 2.

We construct a $7 \times n$ table (Example in Table 2), where each row represents one of the possible reactions, including no reaction, and each column represents the starting configuration's vertices from left to right. For each entry in the table, we see if the reaction exists for that vertex and if the vertex reaches its final state. If both cases are satisfied, place a 1 in the corresponding row, otherwise, place a 0. After all cells are evaluated, we construct a directed graph with edges being directed from column i to column $i+1$ with the following properties for each row entry in column i:

- In final state: edge to every row in column $i+1$ with a 1 except left reaction.
- Left final: edge to every row in column $i+1$ with a 1 except left reaction.
- Left catalytic: edge to every row in column $i+1$ with a 1 except left reaction.
- Left reaction: edge to every row in column $i+1$ with a 1 except left reaction.
- Right final: edge to every row in column $i+1$ with a 1 except left reaction and left final.
- Right catalytic: edge to every row in column $i+1$ with a 1 except left reaction and left catalytic.
- Right reaction: edge only to the row corresponding to left reaction in column $i+1$ if there is a 1.

These edges ensure that no matter which reaction is chosen for a vertex represented by column i, the reaction chosen for the column $i+1$ vertex will be able to perform its reaction either before or after the previous reaction.

Once these edges are defined for every column, the problem is then finding a path from s to t, where s is a vertex that has directed edges to each entry in column 1 and t is a vertex that can be reached from each entry in column n (see Table 2 and Figure 3 for reference). Any path represents a set of rules that can be assigned an ordering to reconfigure all vertices to their final states.

Reaction Type	🟠	⚪	🔵	⚪
In Final State	-	-	-	-
Left	-	1	-	-
Left Catalytic	-	-	-	1
Left Final	-	-	-	-
Right	1	-	-	-
Right Catalytic	-	-	-	-
Right Final	-	-	1	-

Table 2: Turning the example system from Figure 1 into a table of reactions.

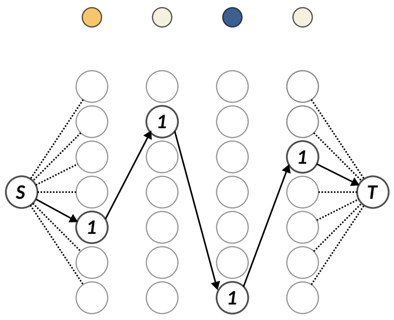


Figure 3: Table 2 as a graph.

Theorem 1 *Reconfiguration in 1-burnout for $1 \times n$ lines is solvable in $O(n + |R|)$ time.*

Proof. We provide proof by induction for the previously described algorithm that solves reconfiguration in $1 \times n$ surfaces. This proof guarantees that any solution from this algorithm constitutes a set of reactions that can be reordered to successfully reconfigure a given initial configuration to its final configuration.

Base case: $n = 2$. Let v_i be the leftmost vertex. Since this vertex does not have a neighbor to its left, there are only 4 reactions we need to consider for this vertex:

1. In final state: vertex v_{i+1} must be in its final state or a left catalytic or left final reaction.
2. Right catalytic: vertex v_{i+1} must be in its final state or a left final reaction.
3. Right final: vertex v_{i+1} must be in its final state or a left catalytic reaction.
4. Right reaction: vertex v_{i+1} must be a left reaction.

If two such reactions exist for each vertex, then a path exists from s to t visiting the vertices in the table that correspond to each reaction. Otherwise, no such path would exist.

Inductive step: let $n = k$. Assume that there is a set of k reactions for vertices $v_1, \ldots, v_k$ that can be reordered to transition all k vertices to their final states.

In order for the reaction chosen for vertex v_{k+1} to be valid, it must not interfere with the k^{th} reaction corresponding to vertex v_k. Consider two cases:

1. Vertex v_k is currently in its final state or reacts with its left neighbor v_{k-1}. Vertex v_{k+1} is never used, so as long as v_{k+1} does not perform a left reaction with v_k, it will not interfere with the k^{th} reaction.
2. Vertex v_k reacts with vertex v_{k+1}. Consider 3 possible reactions for v_k: (1) Right reaction: the only valid reaction for v_{k+1} is a left reaction, (2) Right catalytic: except left or left catalytic, all reactions are valid for v_{k+1}, and (3) Right final: except left or left final, all reactions are valid for v_{k+1}.

If we think of v_k as being column i and v_{k+1} as being column $i+1$, edges are defined from i to $i+1$ in a way that avoid these conflicting reactions. Any other reaction that is chosen for v_{i+1} can always perform its reaction before or after v_i performs its reaction. As a result, any path up to column $i+1$ would represent a set of reactions that can be reordered to transition these $k+1$ vertices to their final states.

Given the initial and final configurations, it takes $O(n)$ time to compare the states. Constructing the table takes $O(|R|)$ time. The path finding algorithm runs in $O(V+E) = O(n+E)$ time. However, the number of edges is a constant factor of the number of vertices, whereas $|R|$ might be exponential in n. Thus, the final runtime for the algorithm is $O(n+|R|)$. □

3.2 2-Burnout Linear Surfaces

Theorem 2 *Reconfiguration of a $1 \times n$ line for surface CRNs with 2-burnout is solvable in $O(n \cdot |S|^2 \cdot |R|^4)$ time.*

Proof. Since we are considering 2-burnout, every cell can only change species twice. This is a cell starting with the initial species, possibly changing to an intermediate species, then finally changing to the target species. It is then possible to track all the possible transitions of a cell in a polynomial sized table. We define the table D with each entry $D(x, s, r_1, r_2)$ being a Boolean indicating if the cells at indices $0, 1, \dots, x$ can reach their target species using reactions r_1 and r_2 on x, and using s as intermediate species for cell x. (Note, r_1 and s may be null if the cell only reacts once to reach the target species.) The reactions are specific with which neighbor the cell reacts with, left or right. This results in $\mathcal{O}(n \cdot |S| \cdot |R|^2)$ cells of the table.

To compute each entry $D(x, s, r_1, r_2)$, we check if r_1 and r_2 are consistent with cell $x-1$. Meaning, if r_1 reacts with the left neighbor, some entry $D(x-1, s', r_1, r_3)$ or $D(x-1, s', r_3, r_1)$ for any s, r_3 must be true. If r_1 is a catalytic reaction, then the species in cell $x-1$ does not change and must be the initial species, intermediate species, or the target species. We must also be careful with the ordering of the reactions. If r_1 or r_2 reacts with the intermediate species s' of the $(x-1)$th cell, then r_1 must be the second reaction for $x-1$. The run time to compute each cell of the table is $\mathcal{O}(|S| \cdot |R|^2)$.

If any $D(n-1, s, r_1, r_2)$ is true, then the answer to reconfiguration is true. □

3.3 Constant Burnout

In this section, we consider the problem of reconfiguration for a surface CRN with n cells with at most k-burnouts on a $1 \times n$ board.

Theorem 3 *There is an $n^{1+k \log h}$-time algorithm for k-burnout degree-h 1D surface CRN reconfiguration, where each species is in at most h rules.*

Proof. We have a divide-and-conquer approach in our algorithm. A brute force method is used to enumerate all the possible transitions for the median position. The problem is split into two independent problems that can be solved independently.

Let p be the position of the median in a 1D surface CRN. We enumerate all the possible ways to burn out the position p at most k times. Since each species is in at most h rules, we have at most h^k combinations about the list of transitions involved by position p. Let $T(n)$ be the running time to solve the reconfiguration problem. We have the recursion $T(n) = h^k(2T(\frac{n}{2}))$. It brings a solution with $T(n) = h^{k \log n} \cdot n = n^{1+k \log h}$. □

4 Non-constant Burnout on a Line

Here, we show that reconfiguration with k-burnout, where k is part of the input, is NP-hard. Without burnout (no bound on state changes), reconfiguration of a $1 \times n$ line is PSPACE-complete [15], but even with a burnout k given in binary, the problem may not be in the class NP since $O(kn)$ possible reactions could occur, which is exponential in $\log k$. This motivates looking at bounds on state changes that are polynomial in n and further motivates the other algorithms in the paper.

Reduction. We reduce from Vertex Cover (VC) by enumerating all vertices and using them as states on a $1 \times n$ line. A state "walks" back and forth choosing a vertex to add to the cover and crossing off instances it finds. Given a graph $G = (V, E)$ where $V = \{1, 2, \dots, n\}$ and an edge $e \in E$ is defined as $e = \{v_i, v_j\}$ for $v_i, v_j \in V$ and $i \neq j$. An edge is listed as two states: 34 meaning an edge between vertices v_3 and v_4. Between any two edges we include a spacing state $-$.

Create the line representing the graph with edges in any order: $BS_0 - e_1 - e_2 - \cdots - e_m - E$, where the B state indicates the beginning of the line, E is the end of the line, and S_0 is a special state indicating no vertices are in the vertex cover. Example: $BS_0 - 34 - 13 - 21 - 14 - E$.

Basically, each edge independently and nondeterministically picks the vertex to cover it with both possible rules. Create rules for all $v_i, v_j \in V$ as $v_i + v_j \to v'_i + x$ and $v_i + v_j \to x + v'_j$ where x is an ignored state and the prime state is the chosen vertex for that edge. The spacing states ensure edges do not affect each other. Example: $3 + 4 \to 3' + x$ and $3 + 4 \to x + 4'$.

The S counting state sweeps back and forth k times to choose a vertex to add to the cover and ignores the other states. The S state takes the first picked vertex and removes all duplicates of it while remembering the count. There is a state S^{vertex}_{count} that exists for each vertex and count up to k. Thus, the rules $S_i + v'_j \to S^j_{i+1} + x$ are added for each vertex and count up to k. Example: $S_0 + 3' \to S^3_1 + x$ is used if v_3 is the first vertex added.

Once a vertex transitions to an S^i state, it ignores everything but v'_i states. Meaning it only swaps states, or "walks" in that direction. Thus, all rules $S^i + A \to A + S^i$ is added for any state X that is not v_i, B, or E. For v_i, $S^i_c + v'_i \to S^i_c + x$.

When a S^j_c vertex is next to the B or E states, it can transition to S_c. The rules $B + S^j_c \to B + S_c$ and $E + S^j_c \to E + S_c$ exist for all vertices v_j. This means we have removed all instances of the chosen vertex and can pick a new vertex for the cover.

This requires $O(kn)$ states to handle counting for each vertex. If k is odd, the final configuration, given a k VC exists, is $B - xx - xx - xx - \cdots - S_kE$. If k is even, then the final configuration is $BS_k - xx - xx - xx - \cdots - E$. S_k can not interact with anything. This requires $k + 1$ burnout.

Theorem 4 *Reconfiguration of a $1 \times n$ configuration in sCRNs with k-burnout is NP-hard, even when $k < n$, and NP-complete as long as k is polynomial in n.*

Proof. Given a VC with graph $G = (V, E)$ and $k \in \mathbb{N}$, we create a surface CRN system with configuration C and rules R as described above. We define the output configuration D based on the number of edges and parity of k as described. G has a VC of size k if and only if C can reach configuration D with burnout $k + 1$. Note that $k \leq n$ as input from VC, so the number of states and rules in the reduction is polynomial.

Given that the graph G has a k vertex cover, in the sCRN system, the only transitions possible at first are for each edge to pick a vertex to cover it. Then the counting state walks across, increases the count and selects the vertex from the first edge, and that state continues walking and removes any other instance of that vertex. In the best case, all locations but the first and last have changed twice. If this continues, and it always adds the correct vertices, then after k passes only x's are left. S_k does not interact with anything, so nothing else transitions. The k passes and the initial choice requires $k + 1$ burnout.

If the sCRN system ends in the output configuration with x's on every edge state, which can only occur if the k passes chose vertices that appeared in the other edges and were crossed out. Thus, every edge correctly chose the right vertex to cover it so that only k different vertices were used. □

5 Extension to 2D Graphs

As an extension to the 1D case, we now consider reconfiguration and 1-reconfiguration for 2D surfaces. In the case of reconfiguration, we study a restricted version of the problem where all reactions are non-catalytic.

Theorem 5 *Reconfiguration in 1-burnout for a planar graph $G = (V, E)$ is solvable in $O(|V|^{1.5} + |R|)$ time if every reaction is non-catalytic.*

Proof. Given a planar graph $G = (V, E)$, construct a subgraph G' from G such that there is an edge between pairs of vertices if there exists a non-catalytic reaction that transitions both vertices to their final states. Run maximum matching on G'. If all vertices are either matched or in their final state, then reconfiguration is possible. Otherwise, reconfiguration is not possible.

Since non-catalytic reactions transition both vertices to their final states, a vertex must be involved in at most one reaction. Edges represent these non-catalytic reactions between two vertices. As a result, limiting a vertex to one reaction is the equivalent of matching each vertex in G' to at most one other vertex it shares an edge with, which is a perfect matching problem. For planar graphs, this can be solved using a maximum matching algorithm. If any unmatched vertex is not in its final state, then reconfiguration is not possible because this vertex is unable to react.

Constructing G' takes $O(V + |R|)$ time. Running the maximum matching algorithm takes $O(V^{1.5})$ time. A last check of G' for any unmatched vertices that are not in their final state takes $O(V)$ time. Therefore, the runtime is $O(V^{1.5} + |R|)$. □

Corollary 6 *Reconfiguration in 1-burnout for general graphs is solvable in $O(V^4 + |R|)$ time if every reaction is non-catalytic, where V is the number of vertices.*

Proof. Proof follows from Theorem 5. Maximum matching on general graphs runs in $O(V^4)$ time. □

5.1 Arbitrary Graphs with 1-Burnout

We now consider surface CRNs that allow catalytic as well as non-catalytic rules. With this additional rule type, we prove the problem of reconfiguration is NP-complete on an arbitrary graph with 1-burnout.

Theorem 7 *Reconfiguration with 1-burnout of an arbitrary surface in surface CRNs is NP-complete.*

Proof. We reduce from the dominating set problem to sCRN reconfiguration with 1-burnout. Let $G = (V, E)$ be an arbitrary graph and k be an integer parameter.

We need to decide if graph G has a dominating set of size k. Note that a subset $U \subseteq V$ is a dominating set of G if each vertex $v \in V - U$ has $(u,v) \in E$ for some $u \in U$ (vertex u dominates v).

Let $v_1, \cdots, v_n$ be the n vertices of G. We design a surface CRN system. For each edge (v_i, v_j) in E, create two rules $v_i + v_j \rightarrow v_i + v_j'$ and $v_i + v_j \rightarrow v_i' + v_j$. We introduce k additional species $u_1, \cdots, u_k$. The target configuration is to let each v_i enter v_i' for $i = 1, \cdots, n$ and each u_t enter u_t'. We set up the rules $u_t + v_j \rightarrow u_t' + v_j'$ for all $t \le k$ and all $j \le n$.

If graph G has a dominating set of size k, the target configuration is reachable. Assume that $v_{i_1}, \cdots, v_{i_k}$ dominate all the vertices in the graph G. For each v_j with $j \in \{1, \cdots, n\} - \{i_1, \cdots, i_k\}$, it can be transformed into v_j' by a rule $v_{i_s} + v_j \rightarrow v_{i_s} + v_j'$. Each v_{i_s} can enter v_{i_s}' by a rule $u_s + v_{i_s} \rightarrow u_s' + v_{i_s}'$. Here, the burnout is 1. Similarly, if the target configuration is reachable, there is a dominating set of size k. If the target configuration is reachable, we have at most $v_{i_1}, \cdots, v_{i_h}$ with $(h \le k)$ such that each v_{i_r} enters v_{i_r}' via the type of rule $u_t + v_{i_r} \rightarrow u_t' + v_{i_r}'$ as there is only one burnout for each v_i and u_j. Clearly, $v_{i_1}, \cdots, v_{i_h}$ dominate all the other vertices in the graph G.

This is a polynomial-time reduction and membership is known from [1]. □

5.2 1-Burnout 1-Reconfiguration

Theorem 8 *1-Reconfiguration in 1-burnout of a $w \times n$ rectangle for surface CRNs is solvable in $O\left(n \cdot (|S||R|)^{2w} \cdot f(w)\right)$ time.*

Proof. We use a dynamic programming approach similar to that in Theorem 2, defining a table D with Boolean entries $D(x, \vec{s}, \vec{r}, \pi)$, where x is a column index, $\vec{s} = [s_1, s_2, \ldots, s_w]$, $\vec{r} = [r_1, r_2, \ldots, r_w]$, and π a permutation of $[1, w]$. Each $s_y \in S$ is a potential final species of cell (x, y), which changes from its initial species into (x, y) due to reaction $r_y \in R$, and π gives the order in which the reactions occur. As before, r_y specifies which of its up-to-four neighboring cells participated in the reaction, and s_y and r_y may be null if the cell never changes species.

Since only one cell (x_t, y_t) of the target configuration is fixed, the top-level of the dynamic program will be column x_t, and it will symmetrically recurse outwards in both directions, with base-cases at both ends. So, for $x < x_t$ $D(x, \vec{s}, \vec{r}, \pi)$ is true if the cells in columns $0, 1, \ldots, x$ can reach a target configuration in which column x reaches species $\vec{s}$ using reactions $\vec{r}$ occurring in order π, for $x > x_t$ we consider columns $x, x+1, \ldots, n-1$ instead, and for $x = x_t$ we consider the entire surface.

To compute $D(x, \vec{s}, \vec{r}, \pi)$, say when $x < x_t$, we search for a smaller subproblem $D(x-1, \vec{s}', \vec{r}', \pi')$ which has value true and $(\vec{r}', \vec{r})$ together are a chain of reactions that actually transform columns $x-1$ and x into species $(\vec{s}', \vec{s})$ from their initial species, given that they must occur in relative orders π' and π. Specifically, we can search each possible interleaving of $\pi(\vec{r})$ and $\pi'(\vec{r}')$, simulate the reactions in that order, and verify that the reactions within these two columns can actually be performed and do result in $(\vec{s}', \vec{s})$. Notably, for reactions between columns $x-2$ and $x-1$, we do not need to validate the species in column $x-2$ because the smaller subproblem already performed that validation, and for reactions between columns x and $x+1$, the species in column $x+1$ are assumed to be validated later in a larger subproblem. For $x > x_t$, the recursion is symmetric.

For top-level subproblems $D(x_t, \vec{s}, \vec{r}, \pi)$, we only consider $\vec{s}$ that include the fixed target species s_{y_t}, and we search for both $D(x_t - 1, \vec{s}', \vec{r}', \pi')$ and $D(x_t + 1, \vec{s}'', \vec{r}'', \pi'')$ and validate between all three columns $x_t - 1, x_t, x_t + 1$ in a similar manner. If any $D(x_t, \vec{s}, \vec{r}, \pi)$ is true, then the answer to 1-reconfiguration is true.

The size of D is $O\left(n \cdot |S|^w \cdot (4|R|)^w \cdot w!\right)$. Computing each entry involves checking $O\left(|S|^w \cdot (4|R|)^w \cdot w!\right)$ subproblems, and each check considers $\binom{2w}{w}$ interleavings of orderings and runs an simulation taking $O(w)$ time. Combined, the total time is $O\left(n \cdot (|S||R|)^{2w} \cdot f(w)\right)$ for a function f only depending on w. Therefore, for constant w, this is polynomial time. □

6 Conclusion

In this paper, we have shown that the reconfiguration problem on $1 \times n$ surface CRNs with k-burnout is in P when $k = 1$ or $k = 2$. To show this, we have given algorithms that output a sequence of reactions to achieve the given configuration. Further, we show that for any $k = \mathcal{O}(1)$, there exists an algorithm that has a polynomial runtime in k. To conclude our investigation of 1-Dimensional surface CRNs, we prove that when the burnout number, k, is part of the input (in unary), the problem of reconfiguration is NP-complete.

Following by exploring 2-Dimensional surface CRNs and showing that a restricted case of 1-burnout reconfiguration can be seen as perfect matching, showing this case of the problem to still be in P. Finishing with a proof that the problem of 1-Reconfiguration in 1-burnout can be solved in polynomial time on a $w \times n$ rectangle when w is constant.

Some of the open questions are then:

- What is the lower bound for a given k-burnout?
- In a rectangle/grid graph, what is the lower/upper bound for k-burnout?
- Most of our complexity is in terms of the size of the surface. Are there interesting results looking at the complexity of other aspects of an sCRN such as states, rules, and burnout?
- We have a direct NP-complete reduction but does there exist an L-reduction for some inapproximability result?

References

[1] R. M. Alaniz, J. Brunner, M. Coulombe, E. D. Demaine, Y. Diomidov, T. Gomez, E. Grizzell, R. Knobel, J. Lynch, A. Rodriguez, R. Schweller, and T. Wylie. Complexity of reconfiguration in surface chemical reaction networks. In *Proc. of the 29th International Conference on DNA Computing and Molecular Programming*, DNA'23, 2023. To appear.

[2] T. Brailovskaya, G. Gowri, S. Yu, and E. Winfree. Reversible computation using swap reactions on a surface. In *Proc. of the International Conference on DNA Computing and Molecular Programming*, DNA'19, pages 174–196. Springer, 2019.

[3] C. Chalk, A. Luchsinger, E. Martinez, R. Schweller, A. Winslow, and T. Wylie. Freezing simulates non-freezing tile automata. In *DNA Computing and Molecular Programming: 24th International Conference, DNA 24, Jinan, China, October 8–12, 2018, Proceedings 24*, pages 155–172. Springer, 2018.

[4] G. Chatterjee, N. Dalchau, R. A. Muscat, A. Phillips, and G. Seelig. A spatially localized architecture for fast and modular DNA computing. *Nature nanotechnology*, 12(9):920–927, 2017.

[5] H.-L. Chen, D. Doty, and D. Soloveichik. Deterministic function computation with chemical reaction networks. *Natural computing*, 13:517–534, 2014.

[6] C. T. Chou. Chemical reaction networks for computing logarithm. *Synthetic Biology*, 2(1):ysx002, Jan. 2017.

[7] S. Clamons, L. Qian, and E. Winfree. Programming and simulating chemical reaction networks on a surface. *Journal of the Royal Society Interface*, 17(166):20190790, 2020.

[8] M. Cook et al. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

[9] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

[10] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. J. Turberfield. DNA walker circuits: computational potential, design, and verification. *Natural Computing*, 14(2):195–211, 2015.

[11] E. Goles, D. Maldonado, P. Montealegre, and M. Ríos-Wilson. On the complexity of asynchronous freezing cellular automata. *Information and Computation*, 281:104764, 2021.

[12] E. Goles, N. Ollinger, and G. Theyssier. Introducing freezing cellular automata. In *Cellular Automata and Discrete Complex Systems, 21st International Workshop (AUTOMATA 2015)*, volume 24, pages 65–73, 2015.

[13] R. A. Muscat, K. Strauss, L. Ceze, and G. Seelig. DNA-based molecular architecture with spatially localized components. *ACM SIGARCH Computer Architecture News*, 41(3):177–188, 2013.

[14] T. Neary and D. Woods. P-completeness of cellular automaton rule 110. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I 33*, pages 132–143. Springer, 2006.

[15] L. Qian and E. Winfree. Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface. In *DNA Computing and Molecular Programming: 20th International Conference, DNA 20, Kyoto, Japan, September 22-26, 2014. Proceedings*, volume 8727, page 114. Springer, 2014.

[16] D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7:615–633, 2008.

[17] G. Theyssier and N. Ollinger. Freezing, bounded-change and convergent cellular automata. *Discrete Mathematics & Theoretical Computer Science*, 24, 2022.

[18] A. J. Thubagere, W. Li, R. F. Johnson, Z. Chen, S. Doroudi, Y. L. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, et al. A cargo-sorting DNA robot. *Science*, 357(6356):eaan6558, 2017.

Catalan Squares and Staircases: Relayering and Repositioning Gray Codes

Emily Downing[*] Stephanie Einstein[†] Elizabeth Hartung[‡] Aaron Williams[§]

Abstract

An n-step staircase can be tiled by n rectangles in C_n ways, where C_n is the n^{th} Catalan number (e.g. , , , , for $C_3 = 5$). We introduce a new Catalan object—*Catalan squares*—by extending each rectangle down and left into an n-by-n square (e.g., to). From this perspective, there are C_n distinct layerings of n squares, where the relative order of ith and kth is concealed when the jth is above them, for any $i < j < k$.

We provide the first Gray codes for these objects. That is, we order the C_n objects so that successive objects differ by a constant amount. More specifically, we provide (a) a relayering Gray code, and (b) a repositioning Gray code, meaning that shapes move to a new layer or are translated to a new position, respectively. We obtain these two Gray codes by working with string-based encodings, including (a) Dyck words (e.g., 110010 for) in cool-lex order, and (b) 231-avoiding permutations (e.g., 132 for) using Algorithm J.

1 Introduction

The *Catalan sequence* $C_0, C_1, C_2, \ldots$ is one of the most well-known sequences in mathematics,

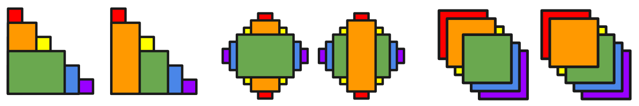
$$1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots (\text{OEIS A000108}[21]).$$

It has natural closed forms and recursive definitions,

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} \tag{1}$$

$$C_n = \prod_{i=0}^{n-i} C_i \cdot C_{n-1-i} \text{ with } C_0 = 1. \tag{2}$$

Catalan objects (i.e., those enumerated by the sequence) are the chameleons of combinatorics. Classic examples include n pairs of balanced parentheses, binary trees with n nodes, triangulations of $(n+2)$-gons, and Stanley's book [22] provides more than 200 examples. In this paper we focus on *Catalan staircases*, which are the C_n tilings of an n-step staircase with n rectangles. We view these objects as being comprised of rectangles of size i-by-$(n-i+1)$ that are layered to create a specific tiling. This view leads to a natural new Catalan object that we refer to as *Catalan squares*. A pair of sample staircases is shown in Figure 1 using each perspective, and Figure 2 clarifies our notion of layers.



(a) Staircases. (b) Centered layers. (c) Catalan squares.

Figure 1: (a) Two Catalan staircases of order $n = 6$ and their representations using (b) centered layers and (c) squares. Center each layer (see Figure 2) to transform (a) to (b). Extend each rectangle down and left into an n-by-n square to transform (a) to (c). The staircases in (a) are the top-right quadrants of (b) and (c).

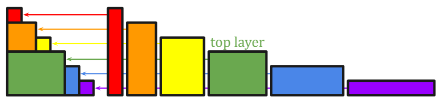


Figure 2: A layered representation views the shapes as i-by-$(n-i+1)$ rectangles that are layered to create distinct tilings. A standard bottom-left alignment is used above, while center-alignments (e.g., Figure 1b) allow all four corners in each shape to be visible. Catalan squares replace the rectangles in the standard bottom-left-aligned view with n-by-n squares.

Our primary goal is to construct Gray codes for these objects for any fixed n. In other words, we order the C_n objects so that successive objects differ by a constant amount. The term *Gray code* is in reference to the *binary reflected Gray code (BRGC)*, named after Frank Gray [7], which orders the n-bit binary strings so that consecutive strings differ in one bit. For example,

$$\text{BRGC}(3) = 00\overline{0}, 0\overline{0}1, 01\overline{1}, \overline{0}10, 11\overline{0}, 1\overline{1}1, 10\overline{1}, \overline{1}00 \tag{3}$$

where overlined bits are flipped to create the next string, including the wrap-around from last $\overline{1}00$ to first 000.

In the context of binary strings, it is clear that flipping a single bit constitutes a small constant-sized change.

[*]Department of Mathematics, Massachusetts College of Liberal Arts, emilylydondowning@gmail.com

[†]Department of Mathematics and Statistics, Mount Holyoke College, einst22s@mtholyoke.edu

[‡]Department of Mathematics, Massachusetts College of Liberal Arts, e.hartung@mcla.edu

[§]Department of Computer Science, Williams College, aw14@williams.edu

This notion is less obvious in the context of Catalan staircases and squares. For this reason, we'll devote some of our attention to representations of these objects, with an emphasis on those that use layers.

We provide a pair of Gray codes that move the shapes (i.e., rectangles or squares) in mutually-exclusive ways.

1. *Relayering staircases or squares.* One or two shapes move to a new layer at the same position.
2. *Repositioning squares.* One square is translated to a new position on the same layer.

Both results are obtained by translating existing string-based Gray codes: Dyck words in cool-lex order and 231-avoiding permutations using Algorithm J.

Vast numbers of Catalan Gray codes have been created over the years, often by manipulating other Catalan objects. For example, binary trees have been created indirectly by making transpositions [15] or adjacent transpositions [16] in the corresponding balanced parentheses or p-sequence [23], or directly using tree rotations [12]. However, this appears to be the first such investigation involving Catalan staircases (and squares).

Sections 2–3 provide background information, and our new results are in Sections 4–5. Section 6 concludes with final remarks; the Appendix adds auxiliary figures.

2 Catalan Objects

This section describes a handful of Catalan objects, starting with string-based objects, and ending with their bijections to Catalan staircases and squares.

2.1 Dyck Words and Balanced Parentheses

A *Dyck word* is a binary string $b_1 b_2 \cdots b_n$ of length $n = 2m$ and *weight* m (i.e., m copies of 1) in which every prefix contains at least as many 1s as 0s. A string of *balanced parentheses* is obtained from a Dyck word by replacing 1s and 0s with (and), respectively. For example, all $C_3 = 5$ such strings with $m = 3$ are below.

$$\begin{array}{ccccc} 101010 & 101100 & 110010 & 110100 & 111000 \\ ()()() & ()(()) & (())() & (()()) & ((())) \end{array} \tag{4}$$

2.2 231-Avoiding Permutations

A permutation $p_1 p_2 \ldots p_n$ *avoids* the pattern 231 unless

$$\exists i, j, k \text{ with } 1 \leq i < j < k \leq n \text{ and } p_k < p_i < p_j. \tag{5}$$

That is, it has no sequence of three symbols with relative order 2, 3, 1. For example, 3412 does not suffice as $34 \cdot 1$ has the forbidden pattern. It is well-known that the number of 231-avoiding permutations of $[n] = \{1, 2, \ldots, n\}$ is C_n. In fact, avoiding any one pattern of length three results in a Catalan object [2]. For example, the $C_4 = 14$ such strings with $n = 4$ are below.

$$\begin{array}{ccccccc} 1234 & 1243 & 1324 & 1423 & 1432 & 2134 & 2143 \\ 3124 & 3214 & 4123 & 4132 & 4213 & 4312 & 4321 \end{array} \tag{6}$$

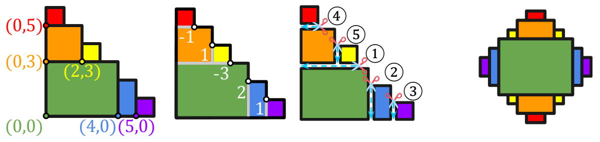
2.3 Catalan Staircases

Recall that a *Catalan staircase* is a tiling of an n-step staircase with n rectangles. We'll view the staircase as having its bottom-left coordinate at the origin $(0, 0)$. The *ith rectangle* is the unique rectangle that touches the *ith corner* $(i, n-i+1)$ for $i = 1, 2, \ldots, n$.

2.3.1 Representations: Rectangle, Line, Slice, Layer

There are several natural geometric representations for Catalan staircases. We consider four categories here, and return to them in Section 6. The first three treat the constituent shapes as they are drawn, and they change dimensions in different tilings. The fourth assumes that the underlying shapes never change, but they are relayered to give the appearance of different tilings.

- A *rectangle representation* directly describes the rectangles. One such representation is a list whose ith entry is the bottom-left co-ordinate of the ith rectangle.
- A *line representation* describes the interior lines. One such representation is a list whose ith entry is the length of a leftward or downward line extending from the point between the ith and $(i+1)$st corner for $1 \leq i < n$. In this case, the length can be measured in units, with negative values for leftward lines.
- A *slice representation* differs from a line representation in that order matters rather than length. More specifically, a Catalan staircase can be created by a sequence of $n-1$ horizontal or vertical cuts, analogous to how a paper cutter can slice a guillotine rectangulation [1].
- A *layer representation* considers the rectangles as having the same perimeter (as in Figure 2) and the tiling is obtained by layering these shapes. Figures 2 and 3d draw the layers with different alignments.



(a) Rectangles. (b) Lines. (c) Slices. (d) Centered layers.

Figure 3: Various geometric representations.

2.4 Catalan Squares

The layer representation of staircases is more purely realized by a new Catalan object. As stated in Section 1, *Catalan squares* can be viewed as modified staircase tilings where each rectangle is extended down and left into an n-by-n square. More directly, we consider n squares of size n-by-n whose top-right corners occupy unique points on the *main diagonal* line segment $(1, n)-(n, 1)$, with the *ith square* being the ith closest to $(1, n)$. We'll often draw the squares in *standard position*, meaning their top-right co-ordinates are $(i, n-i+1)$ for $i = 1, 2, \ldots, n$ (i.e., corner points in a staircase).

Note that the change from rectangles to squares is not simply cosmetic. In particular, Catalan squares support repositioning modifications (see Section 5) while Catalan staircases do not. When discussing Catalan squares we'll differentiate between top/bottom and front/back, with the latter terms referring to the depth of the layer.

2.5 Visibility Property and Depth-Protocol

One may object to Catalan squares being a Catalan object, as there are $n!$ ways to layer n squares. However, some of these choices are *equivalent* in the sense that they appear identical. For example, consider Figure 1c. Note that the relative order of the red and yellow squares is obfuscated by the orange square above them. More generally, the position of the squares along the main diagonal ensures the following *visibility property*:

> The relative order of the ith and kth squares is hidden by the jth square whenever the jth square is in front of them, for all $i < j < k$.

When imagining the squares on different layers, we'll follow this *depth-protocol*: if the jth square is in front of the ith and kth squares, then the kth square is in front of the ith square. This protocol ensures each of the C_n unique objects has one canonical representation among the $n!$ layerings; see Figure 4. The same comments hold for layer representations of Catalan staircases.

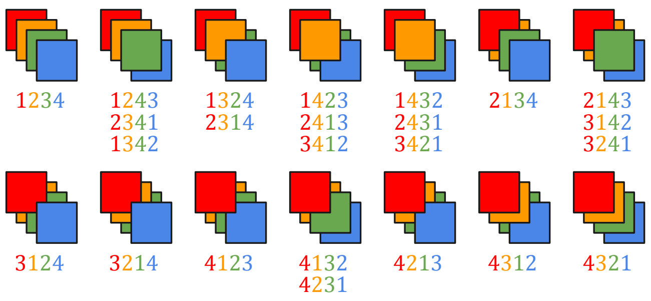


Figure 4: The $4! = 24$ layerings of $n = 4$ squares partitioned into their $C_4 = 14$ equivalence classes. For example, the layerings 4132 and 4231 are equivalent, since the relative order of 1 and 2 is obfuscated by 3 due to the visibility property. In each case, the top permutation is the canonical representative that avoids 231.

2.6 Bijections

This section provides a pair of bijections that are central to Sections 4–5. In both cases, we use the notion of a staircase's *cornerstone*, which is the unique rectangle touching the origin. Cornerstones are then defined recursively for sub-staircases following the familiar recurrence in (2). For example, in Figure 5a, the cornerstone is green, with orange and blue cornerstones for the staircases above it and to its right, respectively.

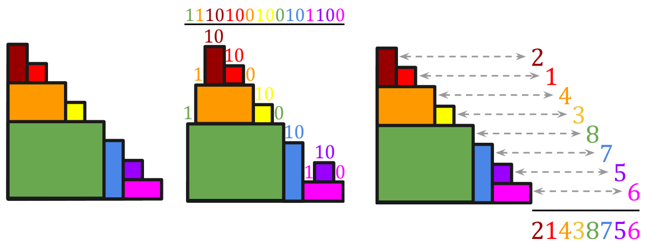
2.6.1 Staircases and Dyck Words

To visualize our first bijection, it is helpful to horizontally center the rectangles above each cornerstone, recursively, rather than use the standard left-alignment. For example, see Figure 5a–5b. We then label the top-left corner of each rectangle with 1 and the top-right corner with 0. A Dyck word is then obtained by reading the bits from left to right, as shown in Figure 5b. We refer to this centered representation as *pyramidal*.

2.6.2 Staircases and 231-Avoiding Permutations

A bijection between 231-avoiding permutations and staircases recursively maps the largest symbol in the permutation to the cornerstone of the staircase. Starting with a 231-avoiding permutation, the largest symbol, n, is the initial cornerstone. Symbols to the left of n correspond to rectangles above the cornerstone, while symbols to the right of n correspond to the rectangles to the right of the cornerstone. We then repeat this process for the sequence of symbols to the left of n and to the right of n: for each sequence, we find the largest symbol, m, which maps to the cornerstone of this subset of rectangles, and shows how many rectangles are above it and to its right within the subset.

The reverse mapping recursively finds the cornerstone and maps it to the largest symbol of the permutation. Rectangles above the cornerstone correspond to values to the left of the largest symbol; rectangles to the right correspond to symbols to the right. If j rectangles are above the cornerstone, then the symbols to the left of the largest symbol, m, are $1, 2, \dots, j$, while the symbols to the right are $j+1, \dots, m-1$. This ensures that the pattern 231 is avoided.



(a) Staircase. (b) Dyck word. (c) Permutation.

Figure 5: Bijectively mapping an (a) Catalan staircase to a (b) Dyck word or (c) 231-avoiding permutation.

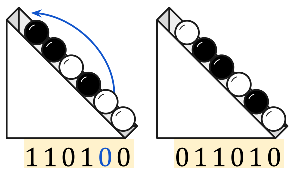
3 Gray Codes

Recall that a *Gray code* is an exhaustive list of some combinatorial object (parameterized by size) in which a constant change occurs from one object to the next. In this section, we'll discuss two specific Gray codes, and how they relate to Catalan objects. For background information on Gray codes, see the classic survey by Savage [20] and the more recent treastise by Mütze [14].

3.1 Cool-lex Order

Cool-lex order is a versatile minimal-change order for strings that is based on the *shift* operation, which removes a symbol and reinserts it elsewhere. More specifically, each shift is a *left-shift*, meaning that the shifted symbol moves to the left. [1] The order was first discovered for *combinations* [19], which are n-bit binary strings in which k of the bits are 1. In other words, they are the incidence vectors of k-subsets of n. The order is notable for its simple successor rule: Shift the bit following the leftmost 01 into the first position. [2]

Cool-lex's successor rule for combinations can be visualized using black and white marbles on a ramp; see Figure 6. Notice that the successor rule rarely changes the rightmost bit. As a result, the order is very similar to co-lexicographic order, from which its name is derived. A generalization of the successor rule generates the permutations of any multiset [24]. In other words, there is a way to shift your way through all arrangements of marbles, regardless of the number of colours.



(a) Before. (b) After.

011100	010110	101001	001011
101100	001110	010101	000111
110100	100110	001101	100011
011010	110010	100101	110001
101010	011001	010011	111000

(c) Full order.

Figure 6: Cool-lex order for combinations with $n = 6$ bits and $k = 3$ copies of 1. (a)-(b) illustrates the successor rule which shifts the bit after the leftmost 01 into the first position; (c) shows the full (cyclic) order where the blue bit is shifted to create the next string.

3.1.1 Bubble Languages and Dyck Words

Dyck words have a simple closure property: If $1^i01\beta$ is a Dyck word, then so too is $1^i10\beta$. In other words, in any Dyck word, the leftmost 01 can be replaced with 10, and the result will be another Dyck word. This modification can be described as 'bubbling' (in reference to bubble sort) the 1 to the left. This property ensures that Dyck words of a given length form a *bubble language*, and that cool-lex order provides a shift Gray code for it [17].

The cool-lex Gray code for Dyck words can be obtained by taking the corresponding sublist of combinations. In other words, if the non-Dyck words are removed from the list of combinations, then the remaining strings will be in a shift Gray code order; see Figure 7a. More importantly, the order of Dyck words also has a simple successor rule. In fact, it differs from the successor rule for combinations in two ways.

1. Bits are shifted into the second position.
2. If the string has a *balanced prefix* 1^i0^i10, then the previous bit is shifted (i.e., 1 is shifted instead of 0).

The successor rule is formally stated in Theorem 1 and is illustrated in Figure 7b.

Theorem 1 **([18])** *The cool-lex order of Dyck words begins with* $101^{n-1}0^{n-1}$ *and the following successor rule generates each successive word where* $p > 0$ *and* $q > 0$.
(a) $1^p0^q11\alpha$ *is followed by* $111^{p-1}0^q1\alpha$.
(b) $1^p0^q10\alpha$ *is followed by* $111^{p-1}0^q0\alpha$ *when* $p = q$.
(c) $1^p0^q10\alpha$ *is followed by* $101^{p-1}0^q1\alpha$ *when* $p > q$.
The next word is obtained by shifting the red bit into the second position; blue bits would be shifted using the successor rule for combinations. The order ends at the final string 1^n0^n, *where none of the above cases applies.*

Cool-lex order for Dyck words of length $n = 2m$ provides a simultaneous Gray code for binary trees with m nodes [18] and ordered trees with $m+1$ nodes [11]. This paper shows that cool-lex's ability to create simultaneous Gray codes extends to more geometric objects.

~~011100~~	~~010110~~	~~101001~~	~~001011~~
101100	~~001110~~	~~010101~~	~~000111~~
110100	~~100110~~	~~001101~~	~~100011~~
~~011010~~	110010	~~100101~~	~~110001~~
101010	~~011001~~	~~010011~~	~~111000~~

(a) Cool-lex order of combinations with all of the non-Dyck words crossed out.

101100
110100
101010
110010
111000

(b) Cool-lex order of Dyck words.

Figure 7: Cool-lex order for Dyck words of length $n = 6$. It is a shift Gray code that is (a) a sublist of Figure 6c, and (b) generated by the successor rule in Theorem 1.

3.2 Greedy Algorithm J

Another prominent Gray code is *plain changes* (or the *Steinhaus–Johnson–Trotter algorithm*) which orders the $n!$ permutations of $[n]$ in one-line notation. The order was used by change ringers in the 1600s [5] before being rediscovered several times in the early 1960s [14]. Its notion of a minimal-change is an *adjacent transposition* (or *swap*) which interchanges two neighboring symbols, as illustrated in Figure 8a.

The order can be understood using local recursion, meaning that each string in Plain(n−1) is expanded to create consecutive strings in Plain(n). More specifically, strings in Plain(n−1) have n inserted in all n possible locations, alternately from right-to-left (zig) and left-to-right (zag), interspersed by the swaps from

[1]Equivalently, a substring is rotated one position to the right.
[2]Special case: If the string has no 01 (e.g., 111000 or 110001), then the rightmost bit is the bit that is shifted.

123	1234	3124	2314
132	1243	3142	2341
312	1423	3412	2431
321	4123	4312	4231
231	4132	4321	4213
213	1432	3421	2413
	1342	3241	2143
	1324	3214	2134

123	1234	3124
132	1243	4312
312	1423	4321
321	4123	3214
213	4132	2134
	1432	2143
	1324	4213

(a) PLAIN(3) and PLAIN(4). (b) Jumps that avoid 231 (e.g., Notice the zig-zagging of n. 321 to 213 and 3124 to 4312).

Figure 8: (a) Plain changes using adjacent transpositions. This is equivalent to Algorithm J's output for all permutations.. (c)–(d) Algorithm J's ordering of 231-avoiding permutations using jumps.

PLAIN(n−1). The order also has a simple greedy interpretation: swap the largest value [25]. [3] More specifically, start with $12 \cdots n$, then extend the order by applying the greediest change that results in a new permutation when applied to the most recent permutation.

A *jump* is a shift (in either direction) where the shifted symbol only moves over smaller symbols. An adjacent transposition is always a jump because we can view the larger symbol as jumping over the smaller symbol. This leads to the following generalized greedy approach known as *Algorithm J*: jump the largest value the shortest possible distance.

3.3 Zig-Zag Languages and 231-Avoidance

The 231-avoiding permutations have a simple inductive property: If α is a valid permutation of $n-1$, then $n \cdot \alpha$ and $\alpha \cdot n$ is a valid permutation for n. As a result, these strings form a *zig-zag language* for any fixed n, and Algorithm J provides a jump Gray code; see Figure 8b.

Theorem 2 ([9]) *Algorithm J provides a jump Gray code for* 231*-avoiding permutations of* $[n]$.

This generalization of plain change order was first announced for permutations avoiding *tame patterns* [8] before being generalized to lattice congruences of the weak order on S_n [9]. Subsequent results considered additional lattice congruences [10], rectangulations [13], elimination trees [4], and acyclic orientations [3]. Our Theorem 4 is a modest contribution to this series of results, with interesting connections to [13].

4 Relayering Gray Code for Staircases and Squares

Now we present our first Gray code for Catalan staircases or squares. Broadly speaking, it is a *relayering Gray code* or a *2-relayering Gray code*, meaning that at most two shapes have their relative depths changed. In other words, if we physically have a layering of n rectangles or squares, then we can create the next layering by pulling out and reinserting at most two of the shapes. Moreover, the depth-protocol will be preserved. For example, Figure 1 shows orange relayered above green.

Theorem 3 *Catalan staircases and Catalan squares have* 2*-relayer Gray codes.*

Proof. Consider Catalan staircases in cool-lex order. In other words, translate each Dyck word in cool-lex order to its corresponding Catalan staircase, as per Section 2.6.1. Figure 9 shows the cool-lex successor rule cases from Theorem 1 with the corresponding staircases; the next is created via one or two relayerings.

More precisely, we can provide a successor rule that maps one staircase into the next. Towards this goal, we introduce some terminology. Two rectangles are *neighbors* if they share any points. A rectangle can have multiple *right neighbors* and/or *top neighbors*, but at most one *left neighbor* and at most one *bottom neighbor*. A rectangle is *thick* if it has height > 1, or equivalently, it has at least one right neighbor; otherwise, it is *thin*.

Let ℓ be the topmost thick rectangle, and s be its bottommost right neighbor. Note that ℓ and s are well-defined except for the final staircase in cool-lex order (which corresponds to 1^n0^n and contains only thin rectangles) where we leave the successor rule undefined. The next staircase is then obtained as follows.

- If s has a top neighbor or no bottom neighbor, then relayer s directly above ℓ. This covers cases (a)–(b).
- Otherwise, relayer s directly above its bottom neighbor b, and the top rectangle t to the front. This covers case (c) and $t = \ell$ is possible.

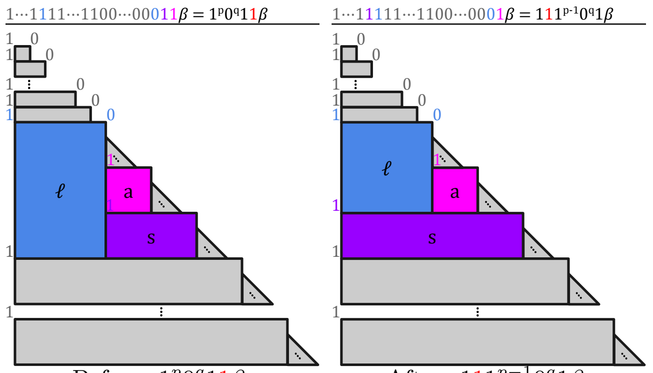
The rule is easily adapted to Catalan squares, and in both settings the depth-protocol is preserved. □

5 Repositioning Gray Code for Catalan Squares

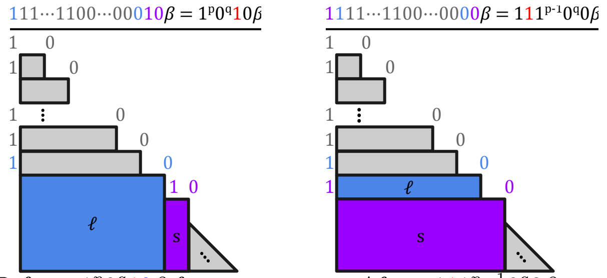
Now we present our second Gray code for Catalan squares. It is a *reposition Gray code*, meaning that one square has its position (but not its depth) changed. In other words, if we have a physical layering of n squares, then we create the next layering by sliding one square to a new position on the main diagonal without changing its depth. This operation is not valid for Catalan staircases as they don't consist of uniform shapes.

Repositions can result in Catalan squares that are not in standard position (see Section 2.4). However, they do satisfy the visibility property (see Section 2.5), and are isomorphic to a configuration in standard position. To realize these changes with physical squares, we'll need to restrict ourselves to repositions that preserve our depth-protocol. These ideas are illustrated in Figure 10.

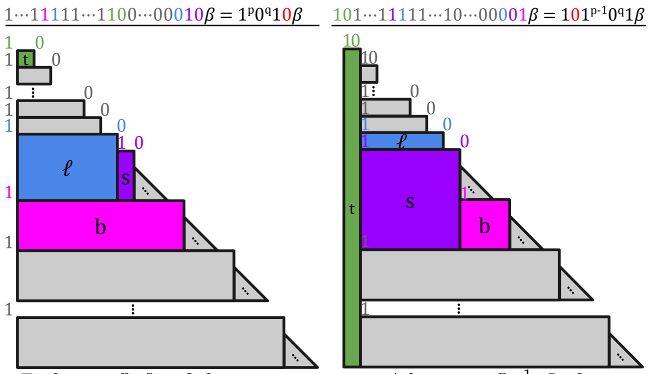
[3]This directive may seem underspecified, since a value can be swapped to the left or right. However, this never ends up being an issue—consider the first two swaps in Figure 8a.



Before: $1^p0^q11\beta$. After: $111^{p-1}0^q1\beta$.
Case (a): Relayer s above ℓ.



Before: $1^p0^q10\beta$ for $p = q$. After: $111^{p-1}0^q0\beta$.
Case (b): Relayer s above ℓ.


1⋯11111⋯1100⋯00010β = 1ᵖ0ᵠ10β
101⋯11111⋯10⋯00001β = 101ᵖ⁻¹0ᵠ1β
t
ℓ
s
b

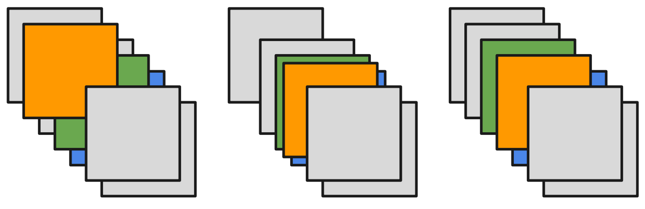
Before: $1^p0^q10\beta$ for $p > q$. After: $101^{p-1}0^q1\beta$.
Case (c): Relayer s above b, then t to the front.

Figure 9: Theorem 3's proof follows Theorem 1's cases. ℓ is the top thick rectangle with no left neighbor; s is ℓ's bottom right neighbor; a is s's left top neighbor (if it exists); b is the bottom neighbor of ℓ and s (if it exists); t is the top rectangle. Gray shapes may or may not exist, with triangles for sub-staircases. Rectangles are thin if drawn as such. See Figure 13 for pyramidal drawings.

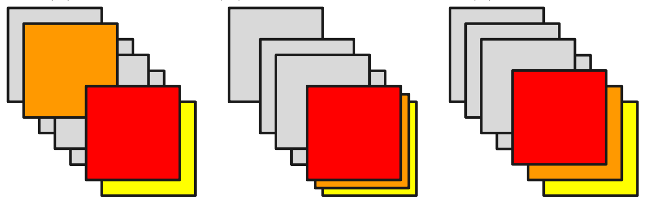
Theorem 4 *Catalan squares have a reposition Gray code.*

Proof. Consider Catalan squares in Algorithm J's order. In other words, translate each 231-avoiding permutation in Algorithm J order to its corresponding Catalan squares, as per the bijection in Section 2.6.2.

A jump of symbol x corresponds to repositioning square x over deeper squares. The jumps performed by Algorithm J do not create permutations with the pattern 231, so they preserve our depth-protocol. □



(a) Before. (b) Valid reposition. (c) Normalize.



(d) Before. (e) Invalid reposition. (f) Normalize.

Figure 10: (a)–(c) A valid reposition where orange moves between green and blue which are below it. (d)–(f) An invalid reposition where orange moves between red and yellow. It is invalid because the depth-protocol has orange below yellow in (d) but above it in (f).

6 Final Remarks

In this paper, we set out to investigate Gray codes for Catalan staircases. Motivated by changes observed using cool-lex order, we developed a layer representation of Catalan staircases, and found that cool-lex order gives a 2-relayering Gray code. The layer representation led to our introduction of Catalan squares, and we found that Algorithm J gives a reposition Gray code.

6.1 Additional Results and Future Work

Theorems 3–4 are existence results, but they have an eye towards efficiency. Cool-lex has efficient ranking and unranking algorithms [18], so our relayering order does as well. Theorem 3's proof also shows how to create each successive staircase. Algorithm J can be generated efficiently using recursion (see Figure 14).

We considered k-ary generalizations of Catalan squares, and how to model them as strings. This led to a pattern avoidance theorem for k-Catalan sequences [26].

More broadly, we are investigating cool-lex order and Algorithm J for other (k-ary) Catalan objects, including bijections with cool-lex order for k-ary Dyck words [6].

6.2 Open Problems

- Do Catalan staircases have 1-relayering Gray codes, or using other representations (see Section 2.3.1)?
- Our orders have $\mathcal{O}(n)$ *distance* (e.g., see bottom-to-top and right-to-left changes in Figures 11–12). Do $\mathcal{O}(1)$ distance Gray codes exist?
- Does Algorithm J have efficient (un)ranking and successor rules for 231-avoiding permutations?
- Is there a C_n formula that mimics Section 2.5?

References

[1] A. Asinowski and T. Mansour. Separable d-permutations and guillotine partitions. *Annals of Combinatorics*, 14:17–43, 2010.

[2] D. Bevan. Permutation patterns: basic definitions and notation. *arXiv preprint arXiv:1506.06673*, 2015.

[3] J. Cardinal, H. P. Hoang, A. Merino, and T. Mütze. Zigzagging through acyclic orientations of chordal graphs and hypergraphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3029–3042. SIAM, 2023.

[4] J. Cardinal, A. Merino, and T. Mütze. Efficient generation of elimination trees and graph associahedra. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2128–2140. SIAM, 2022.

[5] R. Duckworth and F. Stedman. *Tintinnalogia: Or, The Art of Ringing.* Kingsmead Reprints, 1970.

[6] S. Durocher, P. C. Li, D. Mondal, F. Ruskey, and A. Williams. Cool-lex order and k-ary catalan structures. *Journal of Discrete Algorithms*, 16:287–307, 2012.

[7] F. Gray. Pulse code communication. *United States Patent Number 2632058*, 1953.

[8] E. Hartung, H. Hoang, T. Mütze, and A. Williams. Exhaustive generation of pattern-avoiding permutations. In *Proceedings of the 17th International Conference on Permutation Patterns*, pages 81–83, 2019.

[9] E. Hartung, H. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. fundamentals. *Transactions of the American Mathematical Society*, 375(04):2255–2291, 2022.

[10] H. P. Hoang and T. Mütze. Combinatorial generation via permutation languages. II. lattice congruences. *Israel Journal of Mathematics*, 244(1):359–417, 2021.

[11] P. Lapey and A. Williams. Pop & push: Ordered tree iteration in O(1)-time. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[12] J. M. Lucas, D. R. Vanbaronaigien, and F. Ruskey. On rotations and the generation of binary trees. *Journal of Algorithms*, 15(3):343–366, 1993.

[13] A. Merino and T. Mütze. Combinatorial generation via permutation languages. III. rectangulations. *Discrete & Computational Geometry*, 70:51–122, 2023.

[14] T. Mütze. Combinatorial Gray codes-an updated survey. *arXiv preprint arXiv:2202.01280*, 2022.

[15] A. Proskurowski and F. Ruskey. Binary tree Gray codes. *Journal of Algorithms*, 6(2):225–238, 1985.

[16] F. Ruskey and A. Proskurowski. Generating binary trees by transpositions. *Journal of Algorithms*, 11(1):68–84, 1990.

[17] F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex order. *Journal of Combinatorial Theory, Series A*, 119(1):155–169, 2012.

[18] F. Ruskey and A. Williams. Generating balanced parentheses and binary trees by prefix shifts. In *CATS*, volume 8, page 140. Citeseer, 2008.

[19] F. Ruskey and A. Williams. The coolest way to generate combinations. *Discrete Mathematics*, 309(17):5305–5320, 2009.

[20] C. Savage. A survey of combinatorial Gray codes. *SIAM Review*, 39(4):605–629, 1997.

[21] N. J. Sloane et al. The on-line encyclopedia of integer sequences, 2003.

[22] R. P. Stanley. *Catalan Numbers.* Cambridge University Press, 2015.

[23] V. Vajnovszki. Generating a Gray code for P-sequences. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 1(1):31–41, 2002.

[24] A. Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 987–996. SIAM, 2009.

[25] A. Williams. The greedy Gray code algorithm. In *Algorithms and Data Structures: 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings 13*, pages 525–536. Springer, 2013.

[26] A. Williams. Pattern avoidance for k-Catalan sequences. In *Proceedings of the 21st International Conference on Permutation Patterns*, 2023.

Appendix

Figures 11–12 show our $n = 5$ orders. Figures 13–14 show pyramidal drawings, and Algorithm J recursively.

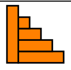
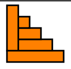
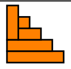
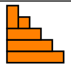
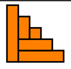
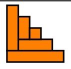
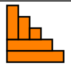
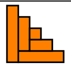
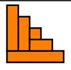
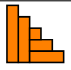
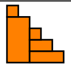
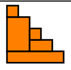
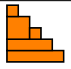
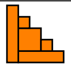
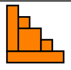
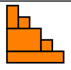
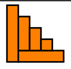
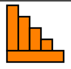
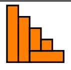
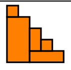
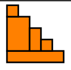
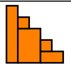
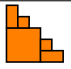
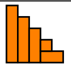
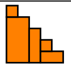
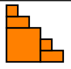
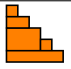
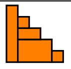
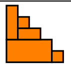
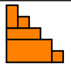
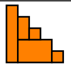
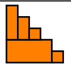
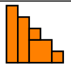
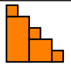
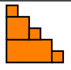
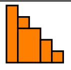
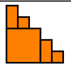
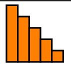
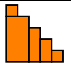
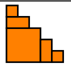
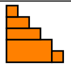
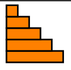
Stairs	Dyck word	Next
	1011110000	(a)
	1101110000	(a)
	1110110000	(a)
	1111010000	(c)
	1011101000	(a)
	1101101000	(a)
	1110101000	(c)
	1011011000	(a)
	1101011000	(c)
	1010111000	(b)
	1100111000	(a)
	1110011000	(a)
	1111001000	(c)
	1011100100	(a)
	1101100100	(a)
	1110100100	(c)
	1011010100	(a)
	1101010100	(c)
	1010110100	(b)
	1100110100	(a)
	1110010100	(c)
	1011001100	(a)
	1101001100	(c)
	1010101100	(b)
	1100101100	(b)
	1110010100	(a)
	1111000100	(c)
	1011100010	(a)
	1101100010	(a)
	1110100010	(c)
	1011010010	(a)
	1101010010	(c)
	1010110010	(b)
	1100110010	(a)
	1110010010	(c)
	1011001010	(a)
	1101001010	(c)
	1010101010	(b)
	1100101010	(b)
	1110001010	(b)
	1111000010	(a)
	1111100000	

Figure 11: The Catalan staircases of order $n = 5$ in cool-lex order. Each Dyck word of length 10 is created using the cool-lex successor rule which left-shifts the red bit into the second position, with cases from Theorem 1. The Dyck words are translated to Catalan staircases using the bijection in Section 2.6.1. The result is a 2-relayering Gray code for the Catalan staircases (or their Catalan square equivalents).

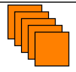
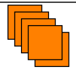
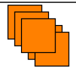
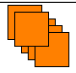
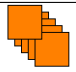
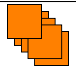
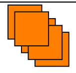
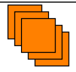
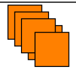
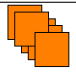
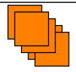
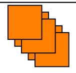
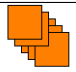
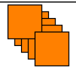
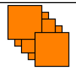
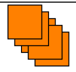
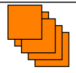
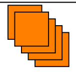
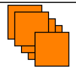
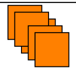
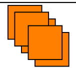
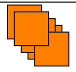
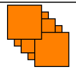
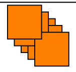
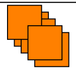
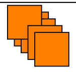
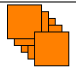
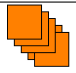
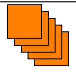
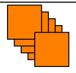
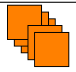
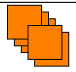
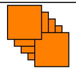
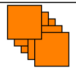
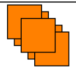
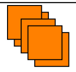
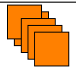
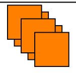
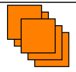
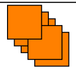
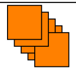
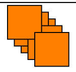
Squares	Permutation
	12345
	12354
	12534
	15234
	51234
	51243
	15243
	12543
	12435
	14235
	15423
	51423
	54123
	41235
	41325
	54132
	51432
	15432
	14325
	13245
	13254
	15324
	51324
	53124
	31254
	31245
	43125
	54312
	54321
	43215
	32145
	32154
	53214
	52134
	21534
	21354
	21345
	21435
	21543
	52143
	54213
	42135

Figure 12: The Catalan squares of order $n = 5$ as generated by Algorithm J. The order is a reposition Gray code, meaning that one square is translated but not raised or lowered (and then the squares are drawn in normal position). Each 231-avoiding permutation is transformed into the next by greedily jumping the largest value the shortest possible distance.

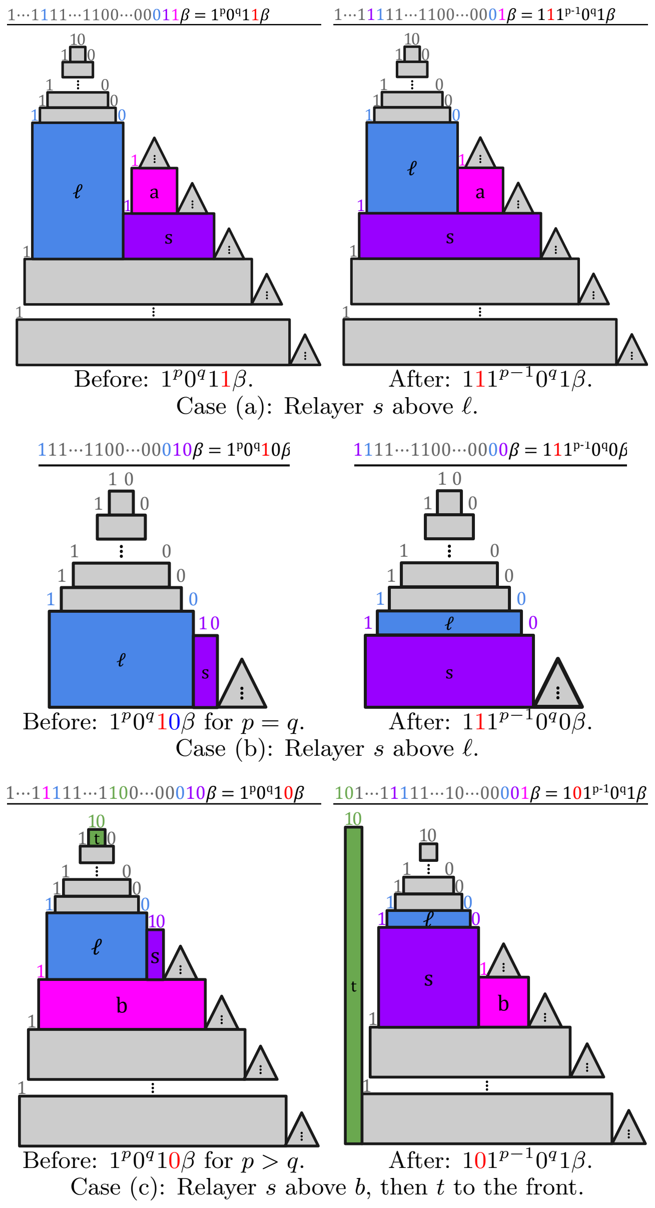

1⋯1111⋯1100⋯00011β = 1ᵖ0ᵠ11β
1⋯11111⋯1100⋯0001β = 111ᵖ⁻¹0ᵠ1β
ℓ
a
s
Before: $1^p0^q11\beta$.
After: $111^{p-1}0^q1\beta$.
Case (a): Relayer s above ℓ.
111⋯1100⋯00010β = 1ᵖ0ᵠ10β
1111⋯1100⋯0000β = 111ᵖ⁻¹0ᵠ0β
Before: $1^p0^q10\beta$ for $p = q$.
After: $111^{p-1}0^q0\beta$.
Case (b): Relayer s above ℓ.
1⋯11111⋯1100⋯00010β = 1ᵖ0ᵠ10β
101⋯11111⋯10⋯00001β = 101ᵖ⁻¹0ᵠ1β
t
b
Before: $1^p0^q10\beta$ for $p > q$.
After: $101^{p-1}0^q1\beta$.
Case (c): Relayer s above b, then t to the front.

Figure 13: A duplication of Figure 9 but with pyramidal drawings. This simplifies the translation to and from Dyck words, at the expense of slightly obfuscating the staircase structure.

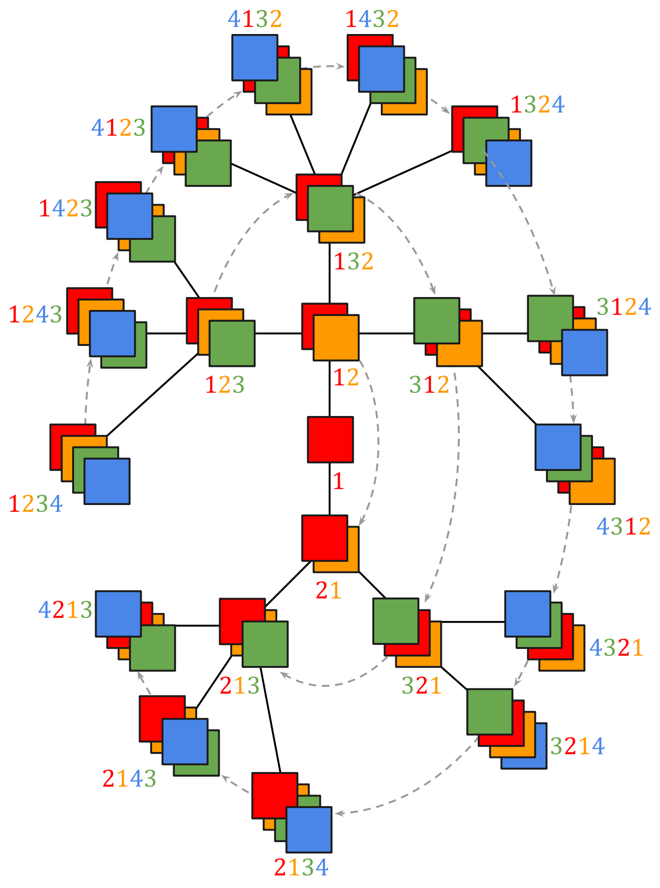


Figure 14: The recursive structure generated by Algorithm J for 231-avoiding permutations and the corresponding Catalan squares for $n = 1, 2, 3, 4$.[4] The Gray codes are obtained by following the gray arrows starting from $12 \cdots n$. Each node's children are obtained by inserting n into the permutation, or repositioning the front square, in all possible ways (i.e., while avoiding the 231 pattern or satisfying the depth-protocol), with the center node 1 as the root. More specifically, nodes at the same depth alternately perform the insertions from left-to-right or right-to-left, thus recreating the familiar zig-zag pattern from Figure 8, which is the hallmark of Algorithm J. This graphic mirrors the tree of generic rectangulations found in [13]. More broadly, this recursive structure creates a jump Gray code for any zig-zag language [9].

[4]The colour scheme used here differs from that in Figure 4.

Computing Representatives of Persistent Homology Generators with a Double Twist

Tuyen Pham[*], Hubert Wagner [†]

Abstract

With the growing availability of efficient tools, persistent homology is becoming a useful methodology in a variety of applications. Significant work has been devoted to implementing tools for persistent homology diagrams; however, computing representative cycles corresponding to each point in the diagram can still be inefficient. To circumvent this problem, we extend the twist algorithm of Chen and Kerber. Our extension is based on a new technique we call *saving*, which supplements their existing *killing* technique. The resulting two-pass strategy can be realized using an existing matrix reduction implementation as a black-box and improves the efficiency of computing representatives of persistent homology generators. We prove the correctness of the new approach and experimentally show its performance.

1 Overview

Persistent homology is a popular methodology for studying geometric and topological information of data. Briefly, as we vary a parameter, persistent homology captures the creation and destruction of topological features present in the data. Typically a persistence diagram is used as a concise geometric-topological descriptor of this evolution. Its usage is becoming popular in various applied fields, including medical imaging [17], astronomy [15], genetics [22] and material science [16].

In many applications it is useful to go beyond this standard descriptor, and study a geometric representation of the captured topological features. More technically, we are referring to the geometry of the representatives of persistent homology generators, which we explain in the next section along with other technicalities; see Figure 1 for an illustration. Visualizing topological information can make topological analysis more intuitive and transparent. Indeed, our work is motivated by a recent project on analyzing neural networks using persistent homology [25]. During this project, we encountered some computational obstacles related to computing representative cycles – and overcome them by creative use of available tools. This enabled visualization and further analysis of important topological features in the data. We share the developed techniques, as they can be applied more generally.

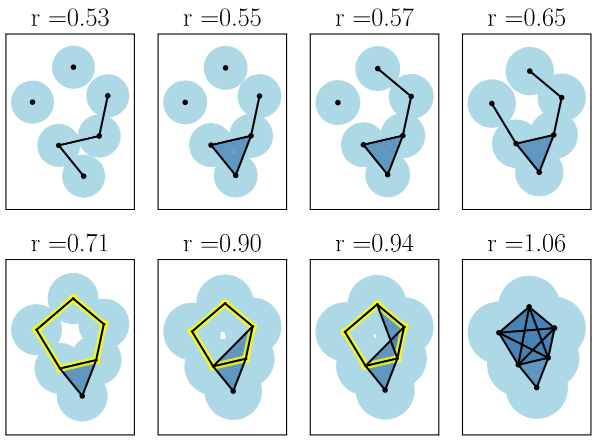


Figure 1: Example of a Vietoris–Rips filtration which approximates the topology of the growing union of disks. Typically information about the birth and death of topological features would be encoded as a persistence diagram. The cycle highlighted in yellow (born at radius 0.71, and destroyed at 1.06) is one representative cycle returned by the algorithm we propose.

These techniques are beneficial because most existing software packages focus on optimizing the computation of persistence diagrams, and not the cycle representatives. Interestingly, the standard algorithm for persistent homology produces cycle representatives (of non-essential classes) with no extra work. On the other hand, applying certain crucial optimizations complicates the situation. As a result, persistence diagrams are typically computed in time approximately linear in the number of input simplices – but computing the representatives can scale quadratically in practical situations. In this work, we aim to close this performance gap by proposing an efficient algorithm for computing representatives of persistent homology generators.

One setting in which our results are particularly useful is low-dimensional skeleta of geometric complexes describing high-dimensional point clouds. In this scenario, it is beneficial [2] to switch to persistent cohomology [9] and apply the crucial killing optimization by Chen and Kerber [8]. While a duality between homol-

*University of Florida, Gainesville; tuyen.pham@ufl.edu
†University of Florida, Gainesville; hwagner@ufl.edu

ogy and cohomology allows us to efficiently compute the persistent homology diagram, we directly obtain only representatives of persistent cohomology generators – and not their easier to interpret and visualize homological counterparts. We offer a simple computational technique which allows us to obtain representatives of persistent homology generators with little extra overhead while benefiting from these crucial optimizations.

Contributions. In short, we propose a more optimistic counterpart of the *killing* technique which we call the *saving* technique. It allows us to generate only a subset of the columns of the boundary matrix, without affecting the results. This technique is part of a new two-pass strategy for computing representatives of non-essential persistent homology classes. In the first pass, we reduce the coboundary matrix using the usual twist algorithm, and generate a subset of the boundary matrix. In the second pass, we reduce the pruned boundary matrix which allows us to retrieve the representatives. We implement this strategy and experimentally show it is typically much faster than reducing the original boundary matrix.

Structure of the paper. In Section 2, we review the usual mathematical background related to persistent homology, trying to make it accessible to audiences with limited exposure to algebraic topology. In Section 3 we briefly review literature on computational aspects of persistent homology. In Section 4 we explain in more details the techniques and algorithms we use in our approach. In Section 5 we explain our approach and then experimentally show its efficiency in Section 6. We conclude the paper in Section 7.

2 Mathematical background

We offer a quick review of the algebraic machinery behind persistent homology [12]. We focus on its common usage in which one computes a sequence of simplicial complexes describing the geometry and topology of a finite point set in $\mathbb{R}^D$. One popular choice is the Vietoris–Rips construction. It allows us to track the birth and death of topological features as a scale parameter is varied. Our main focus is on representatives of persistent homology, which additionally allow us to find geometric representation of topological features.

A k-simplex is the convex hull of $k+1$ affinely independent points in $\mathbb{R}^D$. For $k = 0, 1, 2, 3$, these are vertices, edges, trianges and tetrahedra. A face of a simplex is the convex hull of a subset of its vertices. A face of a simplex σ is called proper if its dimension is one less than the dimension of σ. The boundary of a simplex is the set of its proper faces. By a simplicial complex K we mean collection of simplices such that for every simplex $\sigma \in K$, every face of σ is also in K.

Before we discuss homology groups, we define k-chains as formal sums of simplices with coefficients in $\mathbb{Z}_2$. The k^{th} chain group, $C_k(K)$ is formed by k-chains along with elementwise addition. These chains can be viewed as subsets of simplices in K; the addition reduces to exclusive-difference operation. We define boundary homomorphisms $\partial_k : C_k \to C_{k-1}$ mapping a chain to the sum of the boundaries of the simplices with nonzero coefficients. Crucially, taking the boundary of any chain twice yields the 0 chain. Because of this property, we can define *k-cycles* in $C_k(K)$ as $\ker \partial_n$ and *k-boundaries* in $C_k(K)$ as $\operatorname{im} \partial_{k+1}$. We finally define the degree-k homology group of K as $H_k(K) = \ker \partial_k / \operatorname{im} \partial_{k+1}$. We say that two k-cycles are *homologous* if they belong to the same homology class, namely when one can be formed from the other by adding any k-boundary.

Intuitively, homology group of degree $0, 1, 2$ capture the gaps between components, tunnels and voids of subsets of $\mathbb{R}^3$. Each generator of a k-dimensional homology group can be represented with a k-cycle. In practice, this information can be used to visualize the geometry of each hole present in a dataset.

Persistent homology. A filtration of a simplicial complex K is a sequence of nested simplicial complexes $\emptyset = K_0 \subset K_1 \subset \cdots \subset K_n = K$. For simplicity, we assume K_i is formed by adding a simplex σ_i to K_{i-1}. Upon adding σ_i to K_{i-1}, there are only two possible effects: σ_i either creates a new homology class or destroys an existing one. Depending on the case, we call this simplex a positive simplex or a negative one. For example, adding a 1-simplex (edge) can either connect two existing connected components, or create a new 1-dimensional cycle. Every negative simplex σ_j can be paired with a corresponding positive simplex σ_i with $i < j$. We call this pairing (i, j) an index persistence pair. If σ_i is a positive simplex with no corresponding negative simplex, then σ_i creates an *essential homology class*, namely a homology class of K. In this work we are particularly interested in a computing representatives of non-essential homology classes, namely the ones which are eventually destroyed by a negative simplex.

Boundary matrix. The boundary matrix of the filtration F with n simplices, is a $n \times n$ binary matrix M where $M_{i,j} = 1$ for every pair (σ_i, σ_j) such that σ_i is a proper face of σ_j, and 0 otherwise. The order of the columns and rows of M is induced by the order of the simplices in the filtration. From the boundary matrix, we get the index persistence pairs and representatives of persistent homology generators – which in particular allow us to visualize the changing topology of the filtration. We overview related work in the next section, and provide more details in Section 4.

Reversing the face relationship, we can talk about cofaces, cochains, cocycles, coboundaries, cohomology, and persistent cohomology. We mostly suppress these definitions, and focus on the coboundary matrix, whose columns store proper cofaces of each simplex. We will exploit certain properties of coboundary matrices proven in [9] to compute homological information.

3 Related work

In this section we briefly review literature on computations of (1-parameter) persistent homology. We put emphasise on work which directly inspired this paper.

The standard boundary matrix reduction algorithm for persistent homology is due to Edelsbrunner, Letscher and Zomorodian [13]. The work of de Silva, Morozov and Vejdemo-Johansson [9] proved various dualities between persistent homology and cohomology, which resulted in increased efficiency of the Dionysus library. Chen and Kerber introduced an important *killing* (clearing) optimization. It played a crucial role in the implementation introduced in the PHAT library [2]. This work also experimentally showed the importance of using the clearing optimization along with cohomological computations in the context of skeleta of simplicial complexes. This phenomenon was described in much more detail by Bauer [1], contributing to the efficiency of his popular Ripser package.

Concurrently, various important optimizations [3, 6, 4, 5] were developed in the context of the extensive GUDHI library [19].

Computing *optimal* representatives of homology generators is a computationally hard problem in general [7], but some special cases are more tractable [11, 18]. In this work we do not aim at optimality.

It is worth noting that the algorithmic improvements increase the efficiency by a factor of several thousands times compared to the original algorithm (on the same hardware) [2]. There are also practical situations in which even the optimized algorithms exhibit prohibitively slow scaling, but switching to coboundary matrices and using the twist technique results in linear scaling and overall fast computation. This is an important motivation for our work, and we elaborate on such situations in Section 5.

This efficient behaviour for datasets arising in practice should be contrasted with the worst-case computational complexity, which is $\Theta(n^3)$, where n is the number of simplices in the input filtration. This bound is realized for a synthetic dataset [21]. There exist subcubic algorithms [20], but they remain of theoretical interest. The result of Edelsbrunner and Parsa [14] shows that computing persistence diagrams is as hard as computing the rank of a matrix, leaving little hope for algorithms that would be efficient in the worst case.

We also stress that persistent cohomology and its generating cocycles are powerful tools in their own right [10, 23]. In this paper, however, we reduce their role to computing homology generators – which in some situations are more natural but harder to compute.

Similar techniques to the ones presented in this work were independently devised by Virk and Čufar [24]. Combining with techniques from Ripser [1] allowed for efficient computation of cycle representatives of Vietoris–Rips complexes in Čufar's *Ripserer* software.

4 Existing algorithms

We review selected algorithms for persistent homology computations using matrix reduction techniques. Input to these algorithms is a boundary matrix, typically arising from a simplicial filtration. Output is a reduced matrix, from which persistence pairs and representative cycles (of non-essential classes) can be directly obtained.

More precisely, let M be the boundary matrix of a filtration with the j^{th} column of M denoted M_j. We define the *lowest-one* of the column M_j as $\mathrm{low}(M_j) = \max\{i = 1, \dots, n | M_{i,j} = 1\}$, namely the index of the (visually) lowest nonzero entry in the column M_j.

Standard reduction. The standard matrix reduction algorithm by Edelsbrunner, Letscher and Zomorodian [13] can be summarized as a column-wise Gaussian elimination. The goal is to bring the matrix to a reduced form in which each column has a unique lowest one (or is empty). To this end we perform left-to-right column additions, namely $M_j \leftarrow M_j + M_i$ for $i < j$ when $\mathrm{low}(M_i) = \mathrm{low}(M_j)$. Due to our choice of coefficients, this removes the entry at index $\mathrm{low}(M_j)$ from the column, necessarily decreasing the value of $\mathrm{low}(M_j)$. See Algorithm 1 for pseudocode, and Figure 2 for a simple computational example.

We mention some well-known properties of reduced boundary matrices [12].

Property 1 *Given a reduced matrix M, we extract the index persistence pairs as $(low(M_i), i)$ for each nonzero column M_i.*

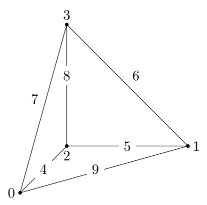
Property 2 *Each zero column in a reduced boundary matrix identifies a positive simplex.*

Property 3 *Each nonzero column of the reduced boundary matrix is a representative of a unique non-essential persistent homology class.*

Twist algorithm. We recall that each simplex in a filtration either creates a single homology class, or destroys one. If it creates a homology class, the corresponding column in the boundary matrix is necessarily zero (i.e. empty when viewed as a set). This fact was exploited by Chen and Kerber [8] to develop the *killing technique*, which zeroes the columns corresponding to positive simplices. Their *twist algorithm* for persistence

Algorithm 1 Standard Boundary Matrix Reduction

```
Require: Boundary matrix $M$ of a simplicial complex filtration with $n$ columns
Ensure: Reduced boundary matrix $M$
1: $L \leftarrow [0, \dots, 0]$ of size $n$
2: for $j = 1, \dots, n$ do
3:     while $M_j \neq 0$ and $L[\mathrm{low}(M_j)] \neq 0$ do
4:         $M_j \leftarrow M_j + M_{L[\mathrm{low}(M_j)]}$
5:     if $M_j \neq 0$ then
6:         $L[\mathrm{low}(M_j)] \leftarrow j$
```



$0 \cdots 3$	4	5	6	7	8	9
$\varnothing$	0	$\cancel{1}$	1	$\cancel{0}$	$\cancel{2}$	$\cancel{0}$
	$\boxed{2}$	$\cancel{2}$	$\boxed{3}$	$\cancel{3}$	$\cancel{3}$	$\cancel{1}$
		0		$\cancel{0}$	$\cancel{1}$	$\varnothing$
		$\boxed{1}$		$\cancel{1}$	$\cancel{2}$	
					$\cancel{0}$	
				$\varnothing$	$\cancel{1}$	
					$\varnothing$	

Figure 2: Standard boundary matrix reduction of a filtration of the 1-skeleton of a tetrahedron. The order in which the vertices and edges are added is determined by the numbers. Intermediate state of each column during reduction is shown.

homology capitalizes on this technique by using the reduced columns corresponding to $(p+1)$-simplices to kill columns corresponding to p-dimensional ones. To this end the algorithm visits columns in *decreasing* order of dimension.

Optimized implementations. Algorithm 2 outlines an implementation of boundary matrix reduction taking into account useful optimizations. The killing optimization [8] is employed in line 12. As shown in [2], it is important to decouple the storage of the columns from the storage and handling of the column being currently reduced. In practice, each column is stored as an array of nonzero indices, but the currently reduced column is represented a data-structure which allows for quick updates and maximum queries. The specialized bit-tree data structure described in [2] is a good choice. Such data structures can be costly to initialized, but this is done only once, in line 2. It is subsequently used in lines 7, 9, 13 at which point it is efficiently cleared.

Coboundary matrix reduction. Persistent homology can be also obtained form a reduced coboundary matrix [9]. More concretely, we consider the anti-transpose of a boundary matrix obtained by mapping each 0-based index i to a dual index $i^* = n - 1 - i$, where n is the total number of simplices. This dual index notation makes it easier to follow computational examples. We remark that in practice it is typically more convenient and faster to generate a coboundary

Algorithm 2 Optimized Boundary Matrix Reduction

```
Require: Boundary matrix $M$ of a simplicial complex filtration of dimension $d$ with $n$ columns
Ensure: Reduced boundary matrix $M$
1: $L \leftarrow [0, \dots, 0]$ of size $n$
2: $C \leftarrow$ data-structure for a column of size $n$
3: for $\delta = d, \dots, 1$ do
4:     for $j = 1, \dots, n$ do
5:         if $j$ corresponds to simplex of dim $\neq \delta$ then
6:             continue
7:         copy $M_j$ to $C$
8:         while $C \neq 0$ and $L[\mathrm{low}(C)] \neq 0$ do
9:             add $M_{L[\mathrm{low}(C)]}$ to $C$
10:        if $C \neq 0$ then
11:            $L[\mathrm{low}(C)] \leftarrow j$
12:            $M_{\mathrm{low}(C)} \leftarrow 0$
13:        move $C$ to $M_j$
```

matrix directly rather than by anti-transposing a previously generated boundary matrix.

Property 4 (Pairing) *If j^* is the lowest one of a column i^* in a reduced coboundary matrix, we obtain an index persistence pair (i^*, j^*) [2, 9].*

We note this is reversed compared to the boundary matrix case: in this case j being the lowest one of i yields (j, i).

The killing technique can be adapted to the case of coboundary matrix as follows [2]. The lowest-one j^* of a nonempty column i^* in the coboundary matrix informs us that column j^* of the coboundary matrix can be killed. In this case, however, simplex σ_{j^*} is of *higher* dimension, which means we should proceed in an *increasing* order of simplex dimensions.

5 Double twist strategy

In this section we first explain certain computational issues arising in practice. We then propose a high-level algorithmic strategy which allows to efficiently compute representatives of persistent homology generators using existing software implementations. To this end, we introduce the counterpart of the *killing* technique that we call the *saving* technique and a two-pass algorithm that we call a *double twist* algorithm.

Simplicial filtrations often arise from Vietoris–Rips or Čech complexes built from a point cloud with n points in $\mathbb{R}^D$. The number of k-simplices can be as large as $\binom{n}{k+1}$. Due to this, we are often restricted to the d-dimensional skeleton, namely the simplices with dimension not exceeding $d < D$. In practice, d is 2, 3, or another small constant. In such cases there are $\Theta(n^{d+1})$ top-dimensional simplices, and these simplices are the most numerous.

The good news is that persistent homology up to degree $d-1$ can be be computed from such a d dimensional skeleton. There is however a subtle computational downside to restricting the complex in this way. In short, the killing technique cannot be used to zero out the columns corresponding to the top-dimensional simplices (since there are no higher dimensional simplices to do the work). In practice, the bulk of work is spent reducing these columns, which makes the killing technique ineffective and often results in prohibitively slow performance [2]. This is not only because these columns are the most numerous – they are also often harder to reduce, since most of them have to be reduced to zero.

Switching to coboundary reduction largely alleviates this problem, as elaborated in [9, 2, 1]. Briefly, the coboundary of each top-dimensional simplex in the input skeleton is empty, so no work is needed. Additionally, the killing technique helps zero-out the columns of penultimate dimension.

However, unlike the columns of the reduced boundary matrix (using column operations), the columns of the reduced coboundary matrix (using column operations) do not contain cycle representatives of homology groups. Instead they capture information about cocycle representatives of cohomology groups. We therefore propose a strategy which allows us to obtain the desired homological information while exploiting the improvement granted by coboundary matrix reduction.

We stress that it is possible to obtain representatives of homological generators directly from the coboundary matrix reduction. This however requires tracking the change of basis matrix [9], which may incur a significant additional cost. Many software implementations, including the PHAT library, avoid this extra cost; the Eirene software by Henselman-Petrusek is a notable exception. One advantage of our technique is that it allows us to use existing, optimized matrix reduction software without any modification. This is not only easier for the user, but also ensures that performance is not inadvertently degraded due to modifications.

Saving technique. The new trick is to use the information contained in the reduced coboundary matrix to prune the boundary matrix. Reducing this pruned boundary matrix yields the desired representatives of homological generators. This technique is used in Algorithm 3 which we call the *double twist* algorithm. Despite involving two passes, experiments in the next section show that this approach is much faster than reducing the original boundary matrix.

More precisely, if a column j^* is the lowest one of a column in the reduced coboundary matrix, we *save* the corresponding simplex $j = n - 1 - j^*$. This is done in lines 4-7 of Algorithm 3. We then construct a boundary matrix – but only the columns corresponding to the previously saved simplices are generated. Dually, all the remaining columns are set to 0 in the constructed boundary matrix. We emphasize that the boundary matrix is not obtained by anti-transposing the reduced coboundary matrix and zeroing out selected columns, which would lose useful information about cycle representatives. See Figure 3 for a computational example using the same input as Figure 2. Next, we prove that the proposed algorithm yields correct results.

Proposition 1 (Saving Works) *The output of the double twist algorithm applied to a filtration F coincides with the reduced matrix B' output by the twist algorithm applied to the boundary matrix B of F.*

Proof. First, Property 4 implies that each saved simplex corresponds to a negative simplex. The remaining ones are therefore positive simplices. Property 2 implies that these columns of B would reduce to zero, so the zero columns are returned by both algorithms. Second, the twist algorithm only adds fully reduced columns to other columns on their right. This implies that the columns corresponding to positive simplices do not affect any of the remaining columns of the reduced matrix. Therefore, the nonzero entries of B are reduced in the same way by both algorithms and the outputs coincide. □

Along with Property 3, the above implies that B' contains representatives of non-essential persistent homology classes of F as columns.

Technicalities. We remark that only a single boundary (or coboundary) matrix is stored at a time. After the first pass, the reduced coboundary matrix can be removed from memory, and we only need to store the information about the saved simplices, which is of negligible size.

Algorithm 3 Double twist strategy

```
Require: Filtration $F$ of a simplicial complex
Ensure: Reduced boundary matrix $B$ containing cycle
    representatives as columns
 1: saved-simplices ← [False, ..., False] of size $n$
 2: $M$ ← coboundary matrix of $F$
 3: reduce $M$ (Alg. 2 but with reversed dimensions)
 4: for $i = 1, \dots n$ do
 5:     if $M_i \neq 0$ then
 6:         $j \leftarrow low(M_i)$
 7:         saved-simplices[$n-1-j$] ← True
 8: delete $M$ from memory
 9: $B$ ← empty boundary matrix with $n$ columns
10: for $i = 1, \dots, n$ do
11:     if saved-simplices[$i$] =True then
12:         $B[i]$ ← boundary of simplex $i$
13: reduce $B$
```

Table 1: Results of computational experiments. Each rows corresponds to different datasets. In particular, the two rightmost columns show the timings of the two matrix reductions in our double twist approach.

Data size	Maximum distance	Number of simplices	Nonzero entries in input (co)boundary matrix	Nonzero entries after 1st pass	Boundary matrix reduction time (naive approach)	Coboundary reduction time (1st pass of proposed alg.)	Pruned boundary reduction time (2nd pass of proposed alg.)
800	1.5	24,134,214	72,195,145	614,492	443.613s	2.445s	0.058s
800	2	85,334,000	255,680,000	958,001	2,292.590s	9.361s	0.157s
1,600	1.3	34,482,186	103,026,427	1, 244,394	950.334s	4.963s	0.125s
1,600	1.5	193,843,549	580,703,170	2,466,432	8,160.460s	23.611s	0.414s
3,200	1	5,767,306	16,991,750	898,505	126.506s	1.391s	0.043s
3,200	1.2	89,682,378	268,002,505	3,101,888	4,753.800s	14.811s	0.279s
6,400	1	44,724,301	132,953,878	3,593,076	2,665.040s	13.482s	0.234s
12,800	1	354,104,851	1,057,473,077	14,396,429	>7,200s	205.768s	2.030s

$$\begin{array}{c|cccc}
9^* \cdots 4^* & 3^* & 2^* & 1^* & 0^* \\
\hline\hline
 & 8^* & 8^* & 9^* & \cancel{9^*} \\
\varnothing & 7^* & 5^* & 6^* & \cancel{7^*} \\
 & \boxed{6^*} & \boxed{4^*} & \boxed{5^*} & \cancel{4^*} \\
 & & & & \vdots \\
 & & & & \varnothing
\end{array}$$

$$\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
[\mathrm{F} & \mathrm{F} & \mathrm{F} & \mathrm{F} & \mathrm{T} & \mathrm{T} & \mathrm{T} & \mathrm{F} & \mathrm{F} & \mathrm{F}\,]
\end{array}$$

$$\begin{array}{c|cccccc}
0 \cdots 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline\hline
 & 0 & \cancel{1} & 1 & & & \\
\varnothing & \boxed{2} & \cancel{2} & \boxed{3} & \varnothing & \varnothing & \varnothing \\
 & & 0 & & & & \\
 & & \boxed{1} & & & &
\end{array}$$

Figure 3: Double twist applied to the same filtration as in Figure 2. After the first pass (top), simplices 4, 5, 6 are *saved* as indicated by the array (middle) representing the saved-simplices variable. After the second pass (bottom) we obtain a matrix identical to the reduced matrix in Figure 2.

6 Experiments

In our experiments we focus on Vietoris–Rips filtrations coming from synthetic data. The aim is to check how the two matrix reductions in the double-twist algorithm scale, compared to a naive strategy in which the boundary matrix is reduced directly. We note that we use a fully optimized twist reduction also for the naive strategy. We also aim to verify that the boundary matrix used in the second pass of our algorithm contains only a small number of nonzero elements.

We implemented our strategy in C++ using the PHAT library. We used a Clang 14.0.3 compiler. The experiments were done on a single core of a 3.5 GHz CPU with 32 GB RAM.

The results are presented in Table 1. Each row corresponds to a single dataset. Each dataset is samples of a 9-dimensional sphere in $\mathbb{R}^{10}$ with a different number of points and radius cutoff. We benchmark both strategies on the 2-skeleton of the Vietoris–Rips filtration for each dataset.

Observations. Analyzing Table 1, we observe that the second pass of the double twist algorithm has negligible impact on the overall execution time (less than 10%). This is not surprising, given the number of nonzero elements in pruned boundary matrix is at least an order of magnitude smaller than compared to the original (co)boundary matrix. Overall, the proposed two-pass algorithm is much faster (up to 200 times) than the naive approach.

The experiments clearly show that there are situations in which using the new double twist strategy is beneficial compared to direct reduction of the boundary matrix. Finally, we add that the results apply also to other situations in which the top-dimensional cells dominate, for example low-dimensional skeleta of high-dimensional Čech, Delaunay or even cubical filtrations.

7 Summary

We proposed a simple algorithmic strategy and showed that – in certain important situations – it significantly speeds up computation of representatives of persistent homology generators. We stress that it does not require implementing a new matrix-reduction algorithm. Instead, any optimized implementation can be used in a black-box fashion, provided it computes index persistence pairs.

We also reiterate that only the representatives of non-essential classes can be obtained from the reduced boundary matrix. While this is actually the more interesting information captured by the persistent homology, we hope that future software packages will support efficient implementation of all representatives. In the meanwhile, however, techniques like the one we proposed serve as a useful workaround.

References

[1] Ulrich Bauer. Ripser: efficient computation of vietoris–rips persistence barcodes. *Journal of Applied and Computational Topology*, 5(3):391–423, 2021.

[2] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. Phat: Persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78:76 – 90, 2017. Algorithms and Software for Computational Topology.

[3] Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70:406–427, 2014.

[4] Jean-Daniel Boissonnat and Siddharth Pritam. Edge collapse and persistence of flag complexes. In *SoCG 2020-36th International Symposium on Computational Geometry*, 2020.

[5] Jean-Daniel Boissonnat, Siddharth Pritam, and Divyansh Pareek. Strong collapse and persistent homology. *Journal of Topology and Analysis*, pages 1–29, 2021.

[6] Jean-Danieland Boissonnat, Tamal K. Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. In *Algorithms – ESA 2013*, pages 695–706, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] Chao Chen and Daniel Freedman. Hardness results for homology localization. *Discrete & Computational Geometry*, 45(3):425–448, 2011.

[8] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European workshop on computational geometry*, volume 11, pages 197–200, 2011.

[9] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in persistent (co)homology. *Inverse Problems*, 27(12):124003, nov 2011.

[10] Vin De Silva and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 227–236, 2009.

[11] Tamal K. Dey. Computing height persistence and homology generators in R^3 efficiently. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, page 2649–2662, USA, 2019. Society for Industrial and Applied Mathematics.

[12] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction.* American Mathematical Soc., 2010.

[13] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Proceedings 41st annual symposium on foundations of computer science*, pages 454–463. IEEE, 2000.

[14] Herbert Edelsbrunner and Salman Parsa. On the computational complexity of betti numbers: reductions from matrix rank. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 152–160. SIAM, 2014.

[15] Sven Heydenreich, Benjamin Brück, and Joachim Harnois-Déraps. Persistent homology in cosmic shear: constraining parameters with topological data analysis. *Astronomy & Astrophysics*, 648:A74, 2021.

[16] Yasuaki Hiraoka, Takenobu Nakamura, Akihiko Hirata, Emerson G Escolar, Kaname Matsue, and Yasumasa Nishiura. Hierarchical structures of amorphous solids characterized by persistent homology. *Proceedings of the National Academy of Sciences*, 113(26):7035–7040, 2016.

[17] Xiaoling Hu, Fuxin Li, Dimitris Samaras, and Chao Chen. Topology-preserving deep image segmentation. *Advances in neural information processing systems*, 32, 2019.

[18] Lu Li, Connor Thompson, Gregory Henselman-Petrusek, Chad Giusti, and Lori Ziegelmeier. Minimal cycle representatives in persistent homology using linear programming: An empirical study with user's guide. *Frontiers in artificial intelligence*, 4:681117, 2021.

[19] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The gudhi library: Simplicial complexes and persistent homology. In *Mathematical Software–ICMS 2014: 4th International Congress, Seoul, South Korea, August 5-9, 2014. Proceedings 4*, pages 167–174. Springer, 2014.

[20] Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the twenty-seventh Annual Symposium on Computational Geometry*, pages 216–225, 2011.

[21] Dmitriy Morozov. Persistence algorithm takes cubic time in worst case. *BioGeometry News, Dept. Comput. Sci., Duke Univ*, 2, 2005.

[22] Raúl Rabadán, Yamina Mohamedi, Udi Rubin, Tim Chu, Adam N Alghalith, Oliver Elliott, Luis Arnés, Santiago Cal, Álvaro J Obaya, Arnold J Levine, et al. Identification of relevant genetic alterations in cancer using topological data analysis. *Nature communications*, 11(1):3808, 2020.

[23] Luis Scoccola, Hitesh Gakhar, Johnathan Bush, Nikolas Schonsheck, Tatum Rask, Ling Zhou, and Jose A Perea. Toroidal coordinates: Decorrelating circular coordinates with lattice reduction. *arXiv preprint arXiv:2212.07201*, 2022.

[24] Matija Čufar and Žiga Virk. Fast computation of persistent homology representatives with involuted persistent homology. *Foundations of Data Science*, (early access), 2023.

[25] Songzhu Zheng, Yikai Zhang, Hubert Wagner, Mayank Goswami, and Chao Chen. Topological detection of trojaned neural networks. *Advances in Neural Information Processing Systems*, 34:17258–17272, 2021.

Approximate Line Segment Nearest Neighbor Search amid Polyhedra in 3-Space

Ovidiu Daescu* Ka Yaw Teo†

Abstract

We consider the problem of finding approximate nearest neighbors for query line segments against a given set of polyhedra with a total of n vertices in $\mathbb{R}^3$. In this problem, for any query line segment s, we wish to quickly report an input polyhedron whose distance to s is at most $(1+\varepsilon)$ times the shortest such distance, for any $\varepsilon > 0$. We present an algorithm that achieves a query time of $O((1/\varepsilon)(n/m^{1/3}) \operatorname{polylog} n + \log n)$ after a preprocessing that takes $O((m/\varepsilon) \operatorname{polylog} n + n^{2+\varepsilon})$ time and space, where $m \in [n, n^3]$ can be specified to yield any desired trade-off between storage and query time. In particular, we require only near-quadratic preprocessing space to answer a query in sub-linear time. In addition, our approach answers exact line segment nearest neighbor queries amid polyhedra with respect to any polyhedral metrics.

1 Introduction

In its most classical sense, a *nearest neighbor search* problem can be defined as follows. Let C be a collection of input geometric objects in $\mathbb{R}^d$, where d is assumed to be a fixed positive integer. Preprocess C so that given a query geometric object q, one can quickly find the object in C nearest to q – that is, the (exact) *nearest neighbor* given by $\mathsf{NN}(q, C) = \arg\min_{c \in C} d(q, c)$, where $d(q, c)$ is the minimum distance between q and c based on some distance metric. In our study, we use the Euclidean norm. An *approximate nearest neighbor* $\mathsf{ANN}(q, C)$ is defined as an object in C such that for a real error parameter $\varepsilon > 0$, $\mathsf{ANN}(q, C) \in \{c \in C | d(q, c) \leq (1+\varepsilon) d(q, \mathsf{NN}(q, C))\}$.

This study focuses on the setting where C is a collection of polyhedra in $\mathbb{R}^3$, and q is a line segment, as formally stated below.

Given a set Π of polyhedra with n vertices, edges, and faces in 3-space, preprocess Π into a data structure such that for any query line segment s, one can efficiently determine the polyhedron in Π closest to s.

We consider the approximate version of the problem, where we wish to return an $\mathsf{ANN}(s, \Pi)$, that is, an input polyhedron whose distance from a query line segment s is at most $(1+\varepsilon)$ times the distance from s to its nearest polyhedron in Π.

2 Related work and motivation

The most basic nearest neighbor search problem, where the input and query objects are points, has been extensively studied for many years, and efficient algorithms exist for solving the problem especially when the dimension d is small [15, 27]. There are efficient solution approaches for high dimensions, provided that one is allowed to report an approximate nearest neighbor, in which case several approximation methods are known and offer trade-offs between the approximation factor, space, and query time (e.g., see [5, 7, 9] and their references).

However, when the input and query objects are more complex than points (e.g., lines, line segments, triangles, polyhedra, and k-flats), the corresponding different variants of nearest neighbor search problem are less understood and remain mostly unresolved. The data structures obtained so far for problems under such settings are typically more costly than those involving points. Note that nearest neighbor searching on non-point objects is at least as hard as that on point objects, for which there are quadratic worst-case lower bounds in exact and approximate settings [8, 22].

On one hand, when the query object is a point, nearest neighbor search problems with the following non-point input objects have been addressed – triangles (and polyhedra in $\mathbb{R}^3$) [23], lines [14, 22, 25], k-flats [4, 10, 24], and line segments [1]. On the other hand, when the input objects are points, the non-point query objects that have been considered include a line [6], a k-flat [28], and a line segment (in $\mathbb{R}^2$) [11, 12, 21, 29]. Among these results, the most closely related recent work to ours is that of Abdelkader and Mount [1], according to which a set of n line segments in $\mathbb{R}^d$ can be preprocessed into a data structure of size $O((n^2/\varepsilon^d)\log(\triangle/\varepsilon))$ such that for any query point, an approximate nearest neighboring line segment can be found in $O(\log(\max\{n, \triangle\}/\varepsilon))$ time, where $\triangle$ is the spread of the input line segments.

*Department of Computer Science, University of Texas at Dallas, ovidiu.daescu@utdallas.edu

†Department of Computer Science, University of Texas at Dallas, ka.teo@utdallas.edu

Daescu and Malik [16] have proposed a data structure for answering exact nearest neighbor search queries when the input objects are (simple) polygons in the plane, and the query object is a line segment. The data structure has a size of $O(m)$, a preprocessing time of $O(m \log n)$, and a query time of $O((n/m^{1/2}) \operatorname{polylog} n)$, for any $n \leq m \leq n^2$. In our study, we are interested in the three-dimensional variant of said problem – specifically, the case where the query object is a line segment, and the input objects are polyhedra in $\mathbb{R}^3$. Just as any other nearest neighbor search problem, our variant has numerous applications in computational geometry, machine learning, and data science, with a particular relevance to path planning problems where one needs to quickly answer collision or clearance queries for non-point objects moving in 3-dimensional space (see Section 5 for further discussion).

3 Our results

In this paper, we provide the first non-trivial algorithm for the approximate line segment nearest neighbor search problem amid polyhedra in three dimensions.

To derive our solution, we first reduce the nearest neighbor search problem to two subproblems – finding i) the nearest neighbor to each endpoint of the query line segment and ii) the nearest orthogonal neighbor to the query line segment (see Section 4 for further description). Our main contribution lies in deriving a solution to the second subproblem.

We approximate the Euclidean distance using an appropriate polyhedral metric. As a result, the first subproblem can be solved efficiently using an approximate Voronoi diagram (see Section 4.1), and the second subproblem can be reduced to a series of edge and face shooting queries (amid edges and points, respectively), which are then addressed using multi-level partition trees (see Section 4.2). As a corollary, we obtain a data structure that finds the *exact* line segment nearest neighbor amid polyhedra in 3-space under a polyhedral metric.

Our data structure has a query time sub-linear in n and uses polynomial preprocessing space and time (see Theorem 3). Specifically, for a $(1+\varepsilon)$ approximation, we obtain a query time of $O((n^{2/3}/\varepsilon) \operatorname{polylog} n)$ with $O(n^{2+\varepsilon})$ preprocessing time and space usage. Unlike the solution for point queries among line segments in 3-space [1], our solution does not depend on the spread of the input objects.

4 Approximate line segment nearest neighbor

In this section, we describe our algorithm for solving the approximate line segment nearest neighbor search problem amid polyhedra in $\mathbb{R}^3$.

We begin by assuming that the faces of the polyhedra in Π are triangular; if that is not the case, we triangulate the faces of the polyhedra in Π and obtain a set of $O(n)$ triangles in 3-space. Let T be the set of $O(n)$ triangular faces of the polyhedra in Π.

Let s denote a query line segment. First, we address the scenario where s intersects a polyhedron of Π. According to a recent work by Ezra and Sharir [20], for any $\alpha > 0$, we can determine if such an intersection exists in time $O(n^{1/2+\alpha})$ after preprocessing T into a data structure of size $O(n^{3/2+\alpha})$ in expected time $O(n^{3/2+\alpha})$. For the rest of the discussion, we assume that s does not intersect any polyhedron in Π.

Let a and b be the two endpoints of the query line segment s. Let H_a (resp. H_b) be the plane containing a (resp. b) and normal to s. Let S_a (resp. S_a) be the closed half-space bounded by H_a (resp. H_b) and not containing s. Define $S_{ab} = \mathbb{R}^3 \setminus (S_a \cup S_b)$.

To find the polyhedron of Π closest to s, we address the following two subproblems individually:

(1) Find the polyhedron in Π closest to each endpoint of s (i.e., a and b).

(2) Find the polyhedron in $\Pi \cap S_{ab}$ closest to s.

We call a polyhedron P in $\Pi \cap S_{ab}$ an *orthogonal neighbor* of s since the closest distance between P and s is given by the shortest orthogonal path from P to s. Note that, of the two polyhedra found in the subproblems above, the one with the shortest distance to s is the polyhedron in Π nearest to s. We now proceed to solve the two subproblems in Section 4.1 and 4.2, respectively.

4.1 Nearest neighbor to each endpoint of s

In this subsection, we consider the following subproblem.

Subproblem 1 *Given a set* Π *of polyhedra with total complexity* n *in* $\mathbb{R}^3$*, preprocess* Π *into a data structure such that for any query point* p*, one can efficiently determine the polyhedron in* Π *closest to* p*.*

We derive an exact solution to Subproblem 1 in the following manner.

Recall that T is the set of $O(n)$ triangular faces of the polyhedra in Π. Subproblem 1 can be reduced to finding the triangle of T nearest to the query point p.

For each triangle τ in T, we define $f_\tau(p)$ to be the Euclidean distance from any point $p \in \mathbb{R}^3$ to τ. Since a point p is given by a 3-tuple (x, y, z), where $x, y, z \in \mathbb{R}$, $f_\tau(p)$ is a piecewise (algebraic) function of three scalar variables x, y, and z.

Let C_T be the collection of functions $\{f_\tau(p) | \tau \in T\}$, and let M_T be the lower envelope of the functions of C_T. Given a query point $p = (x, y, z)$, the triangle $\tau \in$

T nearest to p is the triangle for which function $f_\tau(p)$ attains M_T (in 4-space) at (x, y, z).

Based on a result by Agarwal et al. [3, Theorem 3.3], for any $\alpha > 0$, the lower envelope M_T can be constructed in expected time $O(n^{3+\alpha})$ and stored in a data structure of size $O(n^{3+\alpha})$ such that each nearest triangle searching query described above can be answered in $O(\log^2 n)$ time.

In search of a more efficient data structure, we now choose to resort to approximation as follows.

Let Q to be a (centrally) symmetric convex polytope in $\mathbb{R}^3$. For any two points $p, q \in \mathbb{R}^3$, the polyhedral distance between p and q with respect to Q is defined as $d_Q(p, q) = \sup\{t | q \notin p + tQ\}$. For any $\varepsilon > 0$, there exists a convex polytope Q represented by the intersection of $O(1/\varepsilon)$ half-spaces such that d_Q approximates the Euclidean distance $d(p, q)$ between any two points p and q up to a factor of $(1 + \varepsilon)$ – that is, $d(p, q) \le d_Q(p, q) \le (1 + \varepsilon)d(p, q)$ [19].

We can construct the Voronoi diagram V of Π under a polyhedral distance function (defined with respect to Q) in near-quadratic time [23], and the complexity of V is $O(n^{2+\varepsilon})$. Using V, for any query point p, we can find, in $O(\log n)$ time, a polyhedron in Π whose distance to p is at most $(1 + \varepsilon)$ times the shortest such distance.

Lemma 1 *Let Π be a set of polyhedra with n vertices, edges, and faces in $\mathbb{R}^3$. Let s denote any query line segment, and let p be either endpoint of s.*

i. (Exact) For any $\alpha > 0$, Π can be preprocessed into a data structure of size $O(n^{3+\alpha})$ in expected time $O(n^{3+\alpha})$ such that $\mathsf{NN}(p, \Pi)$ can be reported in $O(\log^2 n)$ time.

ii. (Approximate) For any $\varepsilon > 0$, Π can be preprocessed into a data structure of size $O(n^{2+\varepsilon})$ in time $O(n^{2+\varepsilon})$ such that an $\mathsf{ANN}(p, \Pi)$ can be found in $O(\log n)$ time.

4.2 Nearest orthogonal neighbor to s

The subproblem of interest in this subsection is stated as follows.

Subproblem 2 *Given a set Π of polyhedra of total complexity n in $\mathbb{R}^3$, preprocess Π into a data structure such that for any query line segment s, one can efficiently determine the polyhedron in $\Pi \cap S_{ab}$ closest to s.*

We now proceed to derive an approximate solution to Subproblem 2 – that is, preprocess Π so that an $\mathsf{ANN}(s, \Pi \cap S_{ab})$ can be quickly computed – by using a polyhedral metric.

Recall that Q is a convex polyhedral in $\mathbb{R}^3$ (as previously described in Section 4.1). Let H be a plane orthogonal to s and containing the center of Q. Let Q' be the intersection of Q and H. Note that Q' is a centrally symmetric convex polygon in plane H. Let Q'_a (resp. Q'_b) be a translated copy of Q' centered at a (resp. b). We denote by T the convex polygonal prism with Q'_a and Q'_b as its base faces (Figure 1). Prism T consists of two parallel $O(1/\varepsilon)$-gons (i.e., Q'_a and Q'_b) connected by $O(1/\varepsilon)$ rectangular sides. Note that T is axially symmetrical to s and is defined by $s + tQ'$ for any $t \in \mathbb{R}$.

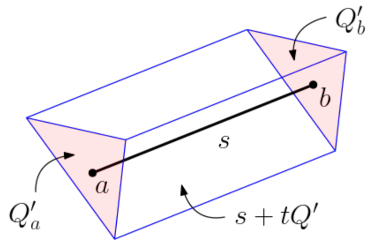


Figure 1: A convex polygonal prism T that is axially symmetrical to line segment s.

Then, the polyhedral distance between s and a polyhedron P in $\Pi \cap S_{ab}$ can be defined as $d_{Q'}(s, P) = \sup\{t | P \cap (s + tQ') = \emptyset\}$. In other words, $d_{Q'}(s, P)$ can be characterized as the smallest expansion factor t such that P makes contact with some face or edge of the axially expanding prism T. Note that for a polyhedron P lying completely outside S_{ab}, we have $d_{Q'}(s, P) = \infty$.

We process each of the $O(1/\varepsilon)$ faces and edges of T for fast face- and edge-shooting queries. There are only two scenarios to be considered:

(A) A shooting face hits a vertex of P.

(B) A shooting edge hits an edge of P.

Scenario A. Briefly, each query shoots a fixed-direction rectangular face from s. The expanding face traces a three-dimensional wedge that emanates from s (Figure 2). We look for the first time at which the expanding wedge hits a vertex of an input polyhedron and return the corresponding polyhedron.

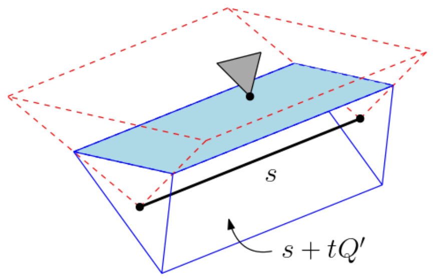


Figure 2: A shooting rectangle in Scenario A.

For each edge e of Q', let $\triangle_e$ denote the three-dimensional wedge defined by $s + \lambda e$, where $\lambda \in \mathbb{R}$ (Figure 3). Notice that $\triangle_e$ is bounded by three rectangular faces. Let c and d denote the two endpoints of edge e. Let $\square_c$ be the rectangle given by $s + \gamma c$, where $\gamma \in \mathbb{R}$. Similarly, let $\square_d$ be the rectangle defined by $s + \gamma d$, where $\gamma \in \mathbb{R}$. Rectangles $\square_c$ and $\square_d$ are bounded by s. Let f_e be the (fixed) shooting rectangular face (i.e., one that is not bordered by s).

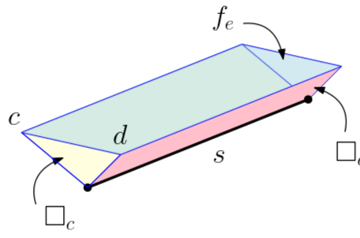


Figure 3: A wedge $\triangle_e$ in Scenario A.

Let V denote the set of $O(n)$ vertices of the polyhedra in Π. For each vertex v of V, there exists a unique scaling factor $\lambda(v) \in \mathbb{R} \cup \infty$ such that v lies in the face f_e of $s + \lambda(v)e$. Note that $\lambda(v) = \infty$ if and only if v does not lie within $\cup_{\lambda(v) \geq 0} s + \lambda(v)e$.

We build a data structure D_e, for each edge e of Q', that answers queries of the following form. Given a query line segment s, find the smallest scaling factor $\lambda^* = \min_{v \in V} \lambda(v)$ and return the corresponding vertex v^* that achieves λ^*.

To begin with, we fix an edge e of Q'. Edge e is bounded by endpoints c and d. We denote by H_c (resp. H_d) the affine plane spanned by $\square_c$ (resp. $\square_d$). Let S_c (resp. S_d) be the closed half-space bounded by H_c (resp. H_d) and containing $\square_d$ (resp. $\square_c$). Note that $\cup_{\lambda \geq 0} s + \lambda e$ is the intersection of four half-spaces S_a, S_b, S_c, and S_d in $\mathbb{R}^3$. Let $S_{abcd} = S_a \cap S_b \cap S_c \cap S_d$. Let u be the (unit) direction vector normal to the shooting face f_e of $\triangle_e$.

We construct a five-level partition tree on V. The first four levels are used to collect, for any query line segment s, the vertices in V that lie within S_{abcd} as the union of a small number of canonical subsets. The fifth level supports queries that ask for the vertex in said canonical subsets that is minimal in direction u.

We follow the standard methodology for constructing multi-level partition trees as detailed in [2, 13, 18, 26]. Each of the first four levels of our partition tree supports half-space range searching queries among the points of V. As described in [2, Theorem 3.4], a d-dimensional half-space range searching query amid a set of n points can be answered in time $O(n/m^{1/d} \operatorname{polylog}(m/n))$ using $O(m)$ space, for any $n \leq m \leq n^d$ (where the preprocessing time is at most a polylogarithmic factor larger than the space).

At the last level, we preprocess each canonical subset S into a data structure so that we can quickly determine the first point in S in direction u. For a set P of n points in d-space where $d \leq 3$, given a query direction vector $u \in \mathbb{R}^d$, we can find the first point of P along direction u in $O(\log n)$ time using $O(n)$ space by constructing the normal diagram of the convex polytope for P and preprocessing it for point-location queries (see Section 6 in [2]).

According to [2, Theorem 6.1], when we construct our multi-level data structure D_e by cascading half-space range searching data structures, each additional level costs at most one logarithmic factor in space and query time. So, the overall preprocessing time and size of data structure D_e are $O(m \operatorname{polylog} n)$, and a query can be answered in time $O(n/m^{1/3} \operatorname{polylog} n)$, for any fixed parameter m where $n \leq m \leq n^3$.

Remark 1 *Alternatively, we can construct a three-level partition tree on V. The first two levels are used to collect, for any given query segment s, the vertices in V that lie within S_{ab} as the union of a small number of canonical subsets. Let L_s be the line supporting s. The third level supports line nearest neighbor searching queries, each of which asks for the vertex in said canonical subsets closest to L_s.*

As given by [17, Corollary 1], a set P of n points in $\mathbb{R}^3$ can be preprocessed using space and time $O(\kappa \operatorname{polylog} n)$ such that for a query line L, the minimum width cylindrical shell enclosing P with central axis L (and thus the exact closest point of P to L) can be computed in $O((n/\kappa^{1/4}) \log n)$, where $n \leq \kappa \leq 4$. By incorporating this result at the third level of our data structure, we can achieve a query time of $O((n/\kappa^{1/4} + n/m^{1/3}) \operatorname{polylog} n)$ with a preprocessing storage of $O((\kappa + m) \operatorname{polylog} n)$. So, when we set $\kappa = m = n^2$, we attain a query time of $O(n^{1/2} \operatorname{polylog} n)$ with a preprocessing space of $O(n^2 \operatorname{polylog} n)$.

Scenario B. There are two types of shooting edges – I) those that are normal to s, and II) those that are parallel to s.

Type I. Each query shoots a fixed line segment normal to s from an endpoint p of s. The expanding line segment traces a two-dimensional wedge that emanates from p (Figure 4). We seek the first time at which the expanding wedge hits an edge of an input polyhedron and return the associated polyhedron.

Note that this scenario has been taken into consideration (indirectly) when solving Subproblem 1, and thus no further action is required.

Type II. Each query shoots a fixed line segment, identical and moving parallel to s, from s. The trajectory of the segment traces a rectangle that emanates

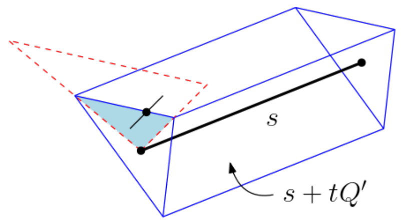


Figure 4: A shooting edge normal to s in Scenario B.

from s (Figure 5). We look for the first time at which the expanding rectangle hits an edge of an input polyhedron and return the corresponding polyhedron.

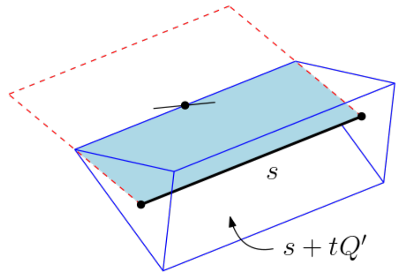


Figure 5: A shooting edge parallel to s in Scenario B.

Formally, let E be the set of $O(n)$ edges of the polyhedra in Π. For each vertex v of Q', we have an expanding rectangle given by $s + \lambda v$, where $\lambda \in \mathbb{R}$. For each edge η of E, there is a unique scaling factor $\lambda(\eta) \in \mathbb{R} \cup \infty$ such that $s + \lambda(\eta)v$ touches η at a point. Note that $\lambda(\eta) = \infty$ if and only if η does not intersect $\cup_{\lambda(\eta)\geq 0} s + \lambda(\eta)v$.

We construct a data structure D_v, for each vertex v of Q', that answers queries of the following form. Given a query line segment s, find the smallest scaling factor $\lambda^* = \min_{\eta\in E} \lambda(\eta)$ and return the corresponding edge η^* that attains λ^*.

First, we fix a vertex v of Q', and let e denote the fixed shooting edge of $s + \lambda v$ parallel to s. For ease of exposition, and without loss of generality, assume that s and e are both located in some plane $z = z_0$; specifically, s is given by $\{(x, y, z)|z = z_0, x = x_0, y_0 - \ell/2 \leq y \leq y_0 + \ell/2\}$, and e is defined by $\{(x, y, z)|z = z_0, x = x_0 + 1, y_0 - \ell/2 \leq y \leq y_0 + \ell/2\}$, where $(x_0, y_0, z_0) \in \mathbb{R}^3$, and ℓ is the length of s.

Suppose that no edge in E is parallel to plane $z = z_0$ (if that is not the case, we apply an infinitesimally small rotation to Q). We can parameterize such an edge η in $\mathbb{R}^3$ using $\{(x, y, z)|x = u_x(\eta)z + v_x(\eta), y = u_y(\eta)z + v_y(\eta), u_z(\eta) \leq z \leq v_z(\eta)\}$, where $u_x(\eta), v_x(\eta), u_y(\eta), v_y(\eta), u_z(\eta), v_z(\eta) \in \mathbb{R}$. Note that η intersects plane $z = z_0$ at point $p_\eta = (u_x(\eta)z_0 + v_x(\eta), y = u_y(\eta)z_0 + v_y(\eta), z_0)$ if $u_z(\eta) \leq z_0 \leq v_z(\eta)$. In which case, p_η lies within the rectangle bounded by s and e when $y_0 - \ell/2 \leq u_y(\eta)z_0 + v_y(\eta) \leq y_0 + \ell/2$. So, we have

$$\begin{aligned} &u_y(\eta)z_0 + v_y(\eta) \geq y_0 - \ell/2,\\ &u_y(\eta)z_0 + v_y(\eta) \leq y_0 + \ell/2,\\ &z_0 - u_z(\eta) \geq 0, \text{and}\\ &z_0 - v_z(\eta) \leq 0. \end{aligned} \tag{1}$$

The constraints (1) are all linear in $u_y(\eta)$, $v_y(\eta)$, $u_z(\eta)$, and $v_z(\eta)$, with coefficients dependent on s. Among all the edges in E that satisfy the inequalities (1), we wish to return the edge that minimizes

$$u_x(\eta)z_0 + v_x(\eta) - x_0, \tag{2}$$

which is linear in $u_x(\eta)$ and $v_x(\eta)$.

To that end, we construct a five-level partition tree on E [2, 13, 18, 26]. The first four levels are to collect, for any query line segment s, the edges in E that satisfy the conditions (1). The fifth level supports linear-programming queries, each of which asks for the edge that attains the minimum of the linear objective function (2).

In detail, we represent each edge $\eta \in E$, which is parameterized by a 6-tuple $(u_x(\eta), v_x(\eta), u_y(\eta), v_y(\eta), u_z(\eta), v_z(\eta))$, using the following four points in $\mathbb{R}^2$:

$$\begin{aligned} p_1(\eta) &= (u_y(\eta), v_y(\eta)),\\ p_2(\eta) &= (1, -u_z(\eta)),\\ p_3(\eta) &= (1, -v_z(\eta)), \text{and}\\ p_4(\eta) &= (u_x(\eta), v_x(\eta)). \end{aligned}$$

Define point sets $P_i = \{p_i(\eta)|\eta \in E\}$, where $1 \leq i \leq 4$. Note that there is a one-to-one correspondence between sets P_i and P_j, where $1 \leq i < j \leq 4$ (and we maintain such mapping through a dictionary). An edge η satisfies the conditions (1) if and only if

i. $p_1(\eta)$ lies above line $xz_0 + y = y_0 - \ell/2$,

ii. $p_1(\eta)$ lies below line $xz_0 + y = y_0 + \ell/2$,

iii. $p_2(\eta)$ lies above line $xz_0 + y = 0$, and

iv. $p_3(\eta)$ lies below line $xz_0 + y = 0$.

In view of the above, the first and second levels of our partition tree are built to support half-plane range searching queries against point set P_1, the third level against P_2, and the fourth level against P_3. Specifically, the first level is a partition tree T over point set P_1, where each node v of T is associated with some canonical subset $P_{1,v}$. For each node v of T, we construct a similar partition tree $T^{(v)}$, as a second-level structure, on the subset $P_{1,v}$. Each node $w \in T^{(v)}$ is associated with a canonical subset $P_{1,v,w} \subset P_{1,v}$. For each node $w \in T^{(v)}$, we construct a partition tree $T^{(w)}$, as a third-level structure, on the subset $P_{2,w} = \{p_2(\eta)|p_1(\eta) \in$

$P_{1,v,w}\}$ of P_2 (put simply, on the subset of points of P_2 whose corresponding points of P_1 are in the canonical subset $P_{1,v,w}$). Each node α of $T^{(w)}$ is associated with a canonical subset $P_{2,w,\alpha} \subset P_{2,w}$. In a similar fashion, we build a partition tree $T^{(\alpha)}$, as a fourth-level structure, on the subset $P_{3,\alpha} = \{p_3(\eta) | p_2(\eta) \in P_{2,w,\alpha}\}$ of P_3 for each node α of $T^{(w)}$. Every node β of $T^{(\alpha)}$ is associated with a canonical subset $P_{3,\alpha,\beta} \subset P_{3,\alpha}$.

Finally, at the fifth level, for each node β of $T^{(\alpha)}$, we preprocess the subset $P_{4,\beta} = \{p_4(\eta) | p_3(\eta) \in P_{3,\alpha,\beta}\}$ of P_4 into a data structure such that for any query vector $u \in \mathbb{R}^2$, the point in $P_{4,\beta}$ that is minimal in direction u can be computed efficiently; in this case, the data structure simply constitutes the convex hull of $P_{4,\beta}$, and such a query can be answered in $O(\log n)$ time.

Putting together, by following the current results on multi-level partition trees and half-space range searching [2, 13, 18, 26], we can construct the data structure D_v using $O(m \operatorname{polylog} n)$ space such that for a query line segment s, the edge $\eta^* \in E$ can be found in time $O(n/m^{1/2} \operatorname{polylog} n)$, where $n \leq m \leq n^2$.

Overall, in order to solve Subproblem 2, we prepare a constant number of face- and edge-shooting data structures, one for each face and edge of T, search with the query line segment s in each of them, and return the closest among the polyhedra output by the queries.

Specifically, in Scenario A, for any $n \leq m \leq n^3$, and for each of the $O(1/\varepsilon)$ faces of T (i.e., each of the $O(1/\varepsilon)$ edges e of Q'), we construct data structure D_e in time $O(m \operatorname{polylog} n)$. Similarly, in Scenario B(II), for any $n \leq m \leq n^2$, and for each of the $O(1/\varepsilon)$ edges of T parallel to s (i.e., each of the $O(1/\varepsilon)$ vertices v of Q'), we build data structure D_v in time $O(m \operatorname{polylog} n)$. This takes a total of $O((m/\varepsilon) \operatorname{polylog} n)$ preprocessing time and storage.

Given a query line segment s, we query each of these data structures with s and return the smallest scaling factor λ (over all said faces and edges of T) and the polyhedron that achieves the minimum. Since, of the two scenarios above, Scenario A has the dominating query time, the total cost of a query is $O((1/\varepsilon)(n/m^{1/3}) \operatorname{polylog} n)$.

Lemma 2 *Let Π be a set of polyhedra with n vertices, edges, and faces in $\mathbb{R}^3$. Let s denote any query line segment. For any $n \leq m \leq n^3$ and $\varepsilon > 0$, Π can be preprocessed into a data structure of size $O((m/\varepsilon) \operatorname{polylog} n)$ in time $O((m/\varepsilon) \operatorname{polylog} n)$ such that an $\mathsf{ANN}(s \cap S_{ab})$ can be found in $O((1/\varepsilon)(n/m^{1/3}) \operatorname{polylog} n)$ time.*

Based on Lemmas 1 and 2, we reach the following conclusion.

Theorem 3 *For any $n \leq m \leq n^3$ and $\varepsilon > 0$, Π can be preprocessed into a data structure of size $O((m/\varepsilon) \operatorname{polylog} n + n^{2+\varepsilon})$ in time $O((m/\varepsilon) \operatorname{polylog} n + n^{2+\varepsilon})$ such that for any query line segment s, one can find an $\mathsf{ANN}(s, \Pi)$ in $O((1/\varepsilon)(n/m^{1/3}) \operatorname{polylog} n + \log n)$ time.*

Particularly, we can find an $\mathsf{ANN}(s, \Pi)$ in time $O((n^{2/3}/\varepsilon) \operatorname{polylog} n)$ using $O(n^{2+\varepsilon})$ space, or in time $O((1/\varepsilon) \operatorname{polylog} n)$ using $O((n^3/\varepsilon) \operatorname{polylog} n)$ space.

5 Concluding remarks

Our nearest neighbor search problem is partly motivated by the following real-world task. In several practical path planning applications, such as emergency intervention, autonomous navigation, and medical treatment planning, (polygonal) paths are suggested by experts in real time, and one is required to quickly answer if the suggested paths satisfy certain constraints, such as a given clearance from the neighboring obstacles. Specifically, given an input set of n polyhedra in 3-space, the goal is to preprocess them so that for any query polygonal path and any $c > 0$, one can efficiently i) report the clearance of the path, and/or ii) determine if the path has a clearance of at least c.

Clearly, query (i) can be answered by finding the exact nearest neighboring input polyhedron, and query (ii) can be easily addressed after determining the path's clearance in query (i). Unfortunately, an exact nearest neighbor search in such a setting is likely to be computationally expensive.

Suppose that we find a $(1 + \varepsilon)$-approximation c_ε to the clearance of the path using the data structure proposed herein. If $c_\varepsilon/(1 + \varepsilon) \geq c$, then we know that the path has a clearance of at least c. Otherwise, the answer is inconclusive, as the path may or may not have a clearance of at least c.

So, the question remains as to whether it is feasible to answer query (ii) definitively without an exact solution to query (i). Note that if we have a decision algorithm for query (ii), we could answer query (i) using optimization methods such as parametric search.

Finally, we leave it to future work to improve the space and time bounds in Theorem 3, possibly by addressing the face and edge shooting queries (reduced from Subproblem 2) using approximation and without resorting to simplex range searching data structures.

References

[1] A. Abdelkader and D. M. Mount. Approximate nearest-neighbor search for line segments. In *37th International Symposium on Computational Geometry*, 2021.

[2] P. K. Agarwal. Simplex range searching and its variants: A review. *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30, 2017.

[3] P. K. Agarwal, B. Aronov, and M. Sharir. Computing envelopes in four dimensions with applications. *SIAM Journal on Computing*, 26(6):1714–1732, 1997.

[4] P. K. Agarwal, N. Rubin, and M. Sharir. Approximate nearest neighbor search amid higher-dimensional flats. In *25th Annual European Symposium on Algorithms*, 2017.

[5] A. Andoni and P. Indyk. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*, pages 1135–1155. Chapman and Hall/CRC, 2017.

[6] A. Andoni, P. Indyk, R. Krauthgamer, and H. L. Nguyên. Approximate line nearest neighbor in high dimensions. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 293–301, 2009.

[7] A. Andoni, P. Indyk, H. L. Nguyên, and I. Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete algorithms*, pages 1018–1028, 2014.

[8] B. Aronov. A lower bound on Voronoi diagram complexity. *Information Processing Letters*, 83(4):183–185, 2002.

[9] S. Arya, G. D. da Fonseca, and D. M. Mount. Optimal approximate polytope membership. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–288, 2017.

[10] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):266–278, 2010.

[11] S. Bespamyatnikh. Computing closest points for segments. *International Journal of Computational Geometry & Applications*, 13(05):419–438, 2003.

[12] S. Bespamyatnikh and J. Snoeyink. Queries with segments in Voronoi diagrams. *Computational Geometry*, 16(1):23–33, 2000.

[13] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47:661–690, 2012.

[14] L. P. Chew, K. Kedem, M. Sharir, B. Tagansky, and E. Welzl. Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. *Journal of Algorithms*, 29(2):238–255, 1998.

[15] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

[16] O. Daescu and H. Malik. Does a robot path have clearance c? In *Proceedings of the 12th International Conference on Combinatorial Optimization and Applications*, pages 509–521, 2018.

[17] O. Daescu and R. Serfling. Extremal point queries with lines and line segments and related problems. *Computational Geometry*, 32(3):223–237, 2005.

[18] D. P. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *Journal of Algorithms*, 8(3):348–361, 1987.

[19] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974.

[20] E. Ezra and M. Sharir. On ray shooting for triangles in 3-space and related problems. *SIAM Journal on Computing*, 51(4):1065–1095, 2022.

[21] P. P. Goswami, S. Das, and S. C. Nandy. Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment. *Computational Geometry*, 29(3):163–175, 2004.

[22] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 94–103, 2001.

[23] V. Koltun and M. Sharir. Polyhedral Voronoi diagrams of polyhedra in three dimensions. *Discrete & Computational Geometry*, 31:83–124, 2004.

[24] A. Magen. Dimensionality reductions in ℓ_2 that preserve volumes and distance to affine spaces. *Discrete & Computational Geometry*, 38(1):139–153, 2007.

[25] S. Mahabadi. Approximate nearest line search in high dimensions. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 337–354, 2014.

[26] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10:157–182, 1993.

[27] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

[28] W. Mulzer, H. L. Nguyên, P. Seiferth, and Y. Stein. Approximate k-flat nearest neighbor search. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, pages 783–792, 2015.

[29] M. Segal and E. Zeitlin. Computing closest and farthest points for a query segment. *Theoretical computer science*, 393(1-3):294–300, 2008.

Universal convex covering problems under affine dihedral group actions[*]

Mook Kwon Jung[†] Sang Duk Yoon[‡] Hee-Kap Ahn[§] Takeshi Tokuyama[¶]

Abstract

We consider the smallest-area universal convex H_k-covering of a set of planar objects, which covers every object in the set allowing the group action of the affine dihedral group $H_k = T \rtimes D_k$ generated by the translation T and the dihedral group D_k. The dihedral group D_k is the group of symmetries of a regular polygon generated by the discrete rotation group Z_k and a reflection. We first classify the smallest-area convex H_k-coverings of the set of all unit segments. Then we show that a suitably positioned equilateral triangle of height 1 is a universal convex H_1-covering of the set S_{c} of all closed curves of length 2. We show that no proper closed subset of the covering is a H_1-covering and the covering is a smallest-area triangle H_1-covering of S_{c}. We conjecture that it is the smallest-area convex H_1-covering of S_{c}. We also show that a suitably positioned equilateral triangle $\triangle_\beta$ of height 0.966 is a universal convex H_2-covering of S_{c}. Finally, we give a universal convex H_3-covering of S_{c} whose area is strictly smaller than that of $\triangle_\beta$.

1 Introduction

Given a (possibly infinite) set S of planar objects and a group G of geometric transformations, a universal G-covering K of S is a region such that every object in S can be contained in K by transforming the object with a suitable transformation $g \in G$. Equivalently, every object of S is contained in $g^{-1}K$ for a suitable transformation $g \in G$. That is,

$$\forall \gamma \in S,\ \exists g \in G \text{ such that } \gamma \subseteq g^{-1}K.$$

A G-covering K of S is *minimal* if no proper closed subset of K is a G-covering of S. We denote the group of planar translations by T and that of planar translations and rotations by TR. Mathematically, $TR = T \rtimes R$ is the semidirect product of T and the rotation group (i.e., the two-dimensional special orthogonal group) $R = SO(2,\mathbb{R})$. We denote O for the orthogonal group $O(2,\mathbb{R})$, which is generated by the rotation group R and a reflection (say, with respect to the x-axis). Our group G is a subgroup of $TO = T \rtimes O$, which is the group generated by T and O. The group TO contains every affine linear transformation for which the shapes of geometric objects are invariant. For simplicity, we often call a universal G-covering a G-covering, or a covering if G is known from the context.

The problem of finding a smallest-area covering is a classical problem in mathematics. In the literature, the cases where $G = T$ or $G = TR$ have been widely studied.

The universal covering problem has attracted many mathematicians. Henri Lebesgue (in his letter to J. Pál in 1914) proposed a problem to find the smallest-area convex TR-covering of all objects of unit diameter (see [7, 4, 11] for its history). Soichi Kakeya considered in 1917 the T-covering of the set S_{seg} of all unit line segments (called *needles*) [15]. Precisely, his formulation is to find the smallest-area region in which a unit-length needle can be turned round, but it is equivalent to the T-covering problem if the covering is convex [3]. Fujiwara conjectured that the equilateral triangle of height 1 is the smallest-area convex T-covering of S_{seg}. The conjecture was affirmatively solved by Pál in 1920 [21]. For the nonconvex variant of the Kakeya problem, Besicovitch [5] gave a construction such that the area can be arbitrarily small, and its variants are widely studied with strong influence on several fields of mathematics [9, 24].

Generalizing Pál's result, for any set of n segments, there is a triangle to be a smallest-area convex T-covering of the set, and the triangle can be computed efficiently in $O(n \log n)$ time [1]. It is further conjectured that the smallest-area convex TR-covering of a family of triangles is a triangle, which is shown to be true for some families [23].

[*]Work by M. K. Jung and H.-K. Ahn were 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 (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)). Work by T. Tokuyama was partially supported by MEXT JSPS Kakenhi 20H04143.

[†]Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea, jmg1032@postech.ac.kr

[‡]Department of Service and Design Engineering, SungShin Women's University, Seoul, Korea, sangduk.yoon@sungshin.ac.kr

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

[¶]Department of Computer Science, School of Engineering, Kwansei Gakuin University, Sanda, Japan, tokuyama@kwansei.ac.jp

The equilateral triangle of height 1 is the smallest-area convex T-covering of the set of all curves of unit length, as well as the unit line segments. In contrast to it, the problem of finding the smallest-area convex TR-covering of the set of all curves of unit length is notoriously difficult. The problem was given by Leo Moser as an open problem in 1966 [17], and it is still unsolved. The best lower bound of the smallest area is 0.21946 [27]. For the best upper bound, Wetzel informally conjectured (formally published in [28]) in 1970 that the 30° circular fan of unit radius, which has an area $\pi/12 \approx 0.2618$, is a convex TR-covering of all unit-length curves, and it was proved by Panraksa and Wichiramala [22]. Recently, the upper bound was improved to 0.260437 [20], but there still remains a substantial gap between the lower and upper bounds.

This problem is known as *Moser's worm problem*, and it has many variants. The history of progress on the topic can be found in an article [18] by William Moser (Leo's younger brother), in Chapter D18 in [8], and in Chapter 11.4 in [7]. It is interesting to find a new variant of Moser's worm problem with a clean mathematical solution.

Let us consider the set S_{c} of all closed curves of length 2. Here, we follow the tradition of previous works on this problem that deals with closed curves of length 2 instead of length 1, since a unit line segment can be considered as a degenerate convex closed curve of length 2. The problem to find a small-area convex covering of S_{c} is known to be an interesting but hard variant of Moser's worm problem, and it remains unsolved for T and TR despite of substantial efforts in the literature [10, 28, 25, 8, 7]. Wichiramala [29] showed that a hexagon obtained by clipping two corners of a rectangle is a convex TR-covering of S_{c}, which has area slightly less than 0.441. It is also shown that any convex TR-covering of S_{c} has area at least 0.39 [12], which has been recently improved to 0.4 [13] with help of computer programs. For convex T-coverings, the smallest area is known to be between 0.620 and 0.657 [7].

There are some works on restricted shapes of covering. Especially, if we consider triangular coverings, Wetzel [25, 26] gave a complete description, and it is shown that an acute triangle with side lengths a, b, c and area X becomes a T-covering (resp. TR-covering) of S_{c} if and only if $2 \le \frac{8X^2}{abc}$ (resp. $2 \le \frac{2\pi X}{a+b+c}$). As a consequence, the equilateral triangle of side length $4/3$ (resp. $\frac{2\sqrt{3}}{\pi}$) is the smallest triangular T-covering (resp. TR-covering) of S_{c}. Unfortunately, their areas are larger than those of the known smallest-area convex coverings.

Finite subgroups of the rotation group $R = SO(2, \mathbb{R})$ are cyclic groups $Z_k = \{e^{2i\pi\sqrt{-1}/k} \mid 0 \le i \le k-1\}$ for $k = 1, 2, \dots$, where $e^{\theta\sqrt{-1}}$ means the rotation of angle θ. The group generated by T and Z_k is denoted by $G_k = T \rtimes Z_k$.

Recently, the convex coverings under the action of the group G_k was investigated by Jung *et al.* [14]. They showed that the smallest-area convex G_2-covering of S_{c} is the equilateral triangle of height 1, whose area is $\frac{\sqrt{3}}{3} \approx 0.577$. They also showed that the equilateral triangle with height $\beta = \cos(\pi/12) \approx 0.966$ is a convex G_4-covering of S_{c}. Its area is $\frac{2\sqrt{3}+3}{12} \approx 0.538675$, and it is conjectured to be the smallest-area convex G_4-covering. If the above conjecture is true, it is a curious phenomenon that the discrete rotations in G_2 and G_4 make the shape of the smallest-area convex covering of S_{c} simple and symmetric compared to the currently known small-area convex T-coverings and TR-coverings.

Among the convex G_3-coverings of S_{c} known so far, the smallest one has area 0.568 [14], and its shape is not a triangle.

There is another type of discrete groups of linear transformations from Z_k for which the shapes of geometric objects are invariant. They are dihedral groups D_k generated by the discrete rotation group Z_k and the reflection with respect to the x-axis. They have order $2k$. As groups, $D_1 \simeq Z_2$, $D_2 \simeq Z_2 \times Z_2$, and $D_3 \simeq S_3$, which is the symmetric group of degree 3. The dihedral group D_3 is also called the A_1-Weyl group as a reflection group. Therefore, it is natural to consider the group $H_k = T \rtimes D_k$ generated by the translation group T and the dihedral group D_k, which we call an *affine dihedral group*. Note that H_k is a subgroup of TO, but not a subgroup of TR.

Our results are as follows.

1. The smallest-area convex H_k-covering of the set S_{seg} of all unit segments is determined for each k.

2. The equilateral triangle of height 1 is an H_1-covering of the set S_{c} of all closed curves of length 2 if and only if it is located so that one of the edges is parallel to the x-axis.

3. The equilateral triangle given above is a minimal convex H_1-covering of S_{c}. It is a smallest-area triangle H_1-covering of S_{c}.

4. The equilateral triangle of height $\beta = \cos(\pi/12)$ is a minimal convex H_2-covering of S_{c} if it is located such that one of its sides has orientation $\pi/4$.

5. The trapezoid obtained by clipping the top corner of the equilateral triangle of height 1 with base parallel to the x-axis is a convex H_3-covering of S_{c}. The area of the covering is strictly smaller than that of the equilateral triangle of height $\beta = \cos(\pi/12)$.

We use elaborate but quite elementary geometric methods to show the results.

Here we introduce the notation and preliminaries. The orientation of a line is the angle swept from the

x-axis in a counterclockwise direction to the line, and it is thus in $[0, \pi)$. The orientation of a segment is the orientation of the line containing the segment. For two points X and Y, we use ℓ_{XY} to denote the line through X and Y. For a compact set U in the plane, we use $|U|$ to denote the area of U. If U is a line segment, then $|U|$ denotes the length of U.

The missing proofs of lemmas and corollaries can be found in the full version.

2 Universal convex coverings of line segments

Since we only consider convex coverings, we say covering for a convex covering from now on unless specifically noticed. We recall a result by Ahn *et al.* [1].

Theorem 1 ([1]) *For any set S of line segments, there exists a triangle that is a smallest-area convex T-covering of S. If S is a finite set and $|S| = n$, such a triangle can be computed in $O(n \log n)$ time.*

The G-orbit of a segment s in S is the set of segments in S to which s can be moved by the elements of a group G. We derive the following corollary from Theorem 1:

Corollary 2 *For any set S of line segments, there is a triangle that is the smallest-area convex H_k-covering.*

The orbit structure of the D_k-action on S_{seg} is not uniform; for example, each of the horizontal and vertical line segments is invariant under the D_2-action and forms a single-element orbit while other orbits have two segments. This contrasts to the Z_k-action, for which the orbit structure is uniform. Consequently, any rotated copy of a G_k-covering of S_{seg} is also a G_k-covering of S_{seg}, but there are cases where a rotated copy of an H_k-covering of S_{seg} is not an H_k-covering of S_{seg}.

Theorem 3 *If k is odd, the smallest area of H_k-coverings of S_{seg} is $\frac{1}{2}\sin\frac{\pi}{2k}$, and it is attained by any triangle $\triangle XYZ$ with horizontal bottom side XY of length 1 and height $\sin\frac{\pi}{2k}$ such that $\frac{\pi}{2} \le \angle X \le \frac{(2k-1)\pi}{2k}$. If k is even, the smallest area of H_k-coverings of S_{seg} is $\frac{1}{2}\sin\frac{\pi}{k}$, and it is attained by any triangle $\triangle XYZ$ with horizontal bottom side XY of length 1 and height $\sin\frac{\pi}{k}$ such that $\frac{\pi}{2} \le \angle X \le \frac{(k-1)\pi}{k}$.*

Proof. The set of orientations of segments in S_{seg} corresponds to the angle interval $[0, \pi)$. First, consider the case where k is odd. Each orbit of Z_k action has exactly k elements. Each of the horizontal and vertical segments has a single orbit in the orbit structure of the action of the reflection with respect to the x-axis while the other segments has 2 orbits in the same orbit structure. The orbit structure of D_k is given by the above combinations, and thus each orbit has at most $2k$ elements.

Let P be an H_k-covering of S_{seg}. For any segment s in S_{seg}, at least one segment in the D_k-orbit of s must be contained in P by translation. Let Y be the set of unit segments that can be contained in P by translation. Consider the smallest angle interval I such that for each segment of orientation θ in Y, $\theta \in I$ or $\theta + \pi \in I$. Observe that I has length at least $\frac{\pi}{2k}$. If the length of I is smaller than $\frac{\pi}{2k}$, the set of orientations of the segments in Y under H_k-action is a proper subset of $[0, \pi)$ since the D_k-orbit of a segment s' of Y has at most $2k$ elements. This contradicts that P is an H_k-covering of S_{seg}. So there are two segments in Y such that their intersection angle $\bar{\theta}$ (the one not larger than $\frac{\pi}{2}$) is not smaller than $\frac{\pi}{2k}$. The convex hull P' of the two segments has area $|P'| = \frac{1}{2}\sin\bar{\theta} \ge \frac{1}{2}\sin\frac{\pi}{2k}$, and $|P| \ge |P'|$. Thus, the smallest area of H_k-coverings is at least $\frac{1}{2}\sin\frac{\pi}{2k}$.

If k is even, each orbit of Z_k action has exactly $\frac{k}{2}$ elements. Thus, each orbit of D_k has at most k elements. The rest is analogous to the odd k case, and the smallest area of H_k-coverings is at least $\frac{1}{2}\sin\frac{\pi}{k}$.

Observe that the triangles given in the theorem are coverings with areas $\frac{1}{2}\sin\frac{\pi}{2k}$ for odd k and $\frac{1}{2}\sin\frac{\pi}{k}$ for even k. □

The triangles obtained by acting elements $h \in H_k$ and $g \in G_2$ on the triangles given in Theorem 3 are also H_k-coverings, since a line segment is invariant with respect to the action of G_2 and the covering condition is invariant with respect to the action of H_k.

Let us compare Theorem 3 with the G_k-coverings of S_{seg} given in [14]. If $k \ge 2$, the smallest area of H_k-coverings is the same as that of the smallest area of G_{2k}-coverings. The smallest area of H_1-coverings is $\frac{1}{2}$, which is the same as the smallest area of G_4-coverings. In contrast, the smallest-area G_2-covering (that is the same as the smallest-area T-covering) of S_{seg} is the equilateral triangle of area $\frac{1}{\sqrt{3}}$.

3 Universal convex H_1-coverings of S_{c}

In this section, we consider H_1-coverings of the set S_{c} of all closed curves of length 2. First, we recall known results mentioned in the introduction.

3.1 The smallest-area covering and related results

Theorem 4 (Pal's theorem) *The equilateral triangle of height 1 is the smallest-area (convex) T-covering of the set of all unit line segments.*

Corollary 5 *The area of a G_2-covering of S_{c} is at least $1/\sqrt{3}$.*

Proof. Observe that all unit line segments are in S_{c}, and line segments are stable under the action of rotation by π. Thus, any convex G_2-covering of S_{c} must be a T-covering of all unit line segments, and the corollary follows from Theorem 4 (Pal's theorem). □

It is known that the above lower bound is tight.

Theorem 6 (Jung *et al.* [14]) *The equilateral triangle of height* 1 *is the smallest-area G_2-covering of S_{c}.*

Lemma 7 *Suppose that a region P is symmetric with respect to the y-axis. Then the following holds.*

- *For an odd k, P is an H_k-covering of S_{c} if and only if it is a G_{2k}-covering of S_{c}.*
- *For an even k, P is an H_k-covering of S_{c} if and only if it is a G_k-covering of S_{c}.*

Proof. Let h be the reflection with respect to the x-axis, and let g be the rotation of angle π about the origin. From the assumption that P is symmetric with respect to the y-axis, $h \cdot P$ is a translation of $g \cdot P$. If k is even, G_k contains g. Hence the H_k-orbit of P is the same as the G_k-orbit of P, and we have the second statement. If k is odd, the group generated by G_k and g is G_{2k}, and we have the first statement. □

3.2 H_1-coverings of S_{c}

Let $\triangle_1$ be an equilateral triangle of height 1 whose bottom side is horizontal.

Theorem 8 *The equilateral triangle $\triangle_1$ is an H_1-covering of S_{c}. Moreover, it is the smallest-area H_1-covering among all H_1-coverings of S_{c} that are convex and symmetric to the y-axis.*

Proof. The first statement follows immediately from Theorem 6 and Lemma 7. Consider a H_1-covering P that is convex and symmetric with respect to the y-axis. By Lemma 7, P is G_2-covering. By Theorem 6, $|\triangle_1| \leq |P|$. Thus, the second statement also holds. □

Corollary 9 *Any closed curve of length* 2 *that is symmetric with respect to a line of orientation* 0*, $\pi/3$ or $2\pi/3$ is contained in $\triangle_1$ by translation.*

This corollary complements the fact that any centrally-symmetric closed curve of length 2 can be contained in $\triangle_1$ by translation [14]. However, $\triangle_1$ is not the smallest-area T-covering of the set of the closed curves of length 2 that are symmetric about the x-axis, since a square with a unit length horizontal diagonal is a T-covering of the set. The area of the square is $\frac{1}{2}$, and thus smaller than that of $\triangle_1$. Thus, the H_1-covering problem and the T-covering problem of D_1-invariant objects have different solutions.

Theorem 10 *Let T_L be an equilateral triangle of perimeter* 2 *such that it has a vertical side and its opposite corner lies to the right. Let T_R be a copy of T_L rotated by π. The equilateral triangle $\triangle_1$ and its reflected image about the x-axis are the only convex H_1-coverings of T_L and T_R among the rotated copies of $\triangle_1$ about the origin.*

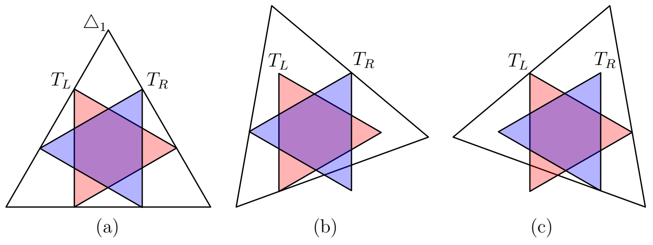


Figure 1: (a) Translates of T_L and T_R that are contained in $\triangle_1$. (b) No translate of T_R can be contained in a rotated copy of $\triangle_1$ by θ with $0 < \theta < \pi/3$. (c) No translate of T_L can be contained in a rotated copy of $\triangle_1$ by θ with $-\pi/3 < \theta < 0$.

Proof. Observe that there are translates of T_L and T_R that are contained in $\triangle_1$. See Figure 1 (a). Observe that both T_L and T_R are symmetric with respect to a horizontal line. If a rotated copy $\triangle_\theta$ of $\triangle_1$ by θ is a H_1-covering of T_L and T_R, there are translates of T_L and T_R that are contained in $\triangle_\theta$. Observe that no translate of T_R is contained in $\triangle_\theta$ with $0 < \theta < \pi/3$ and no translate of T_L is contained in $\triangle_\theta$ with $-\pi/3 < \theta < 0$, as shown in Figure 1 (b) and (c). $\triangle_1$ is invariant under rotation by $2\pi/3$. The rotation by $\pi/3$ of $\triangle_1$ is equivalent to $g \cdot \triangle_1$, where g is the reflection with respect to the x-axis. Thus, $\triangle_1$ and $g \cdot \triangle_1$ are the only convex H_1-coverings of T_L and T_R among the rotated copies of $\triangle_1$. □

3.3 The minimality of the H_1-covering $\triangle_1$

One may wonder whether we may remove some part of $\triangle_1$ to obtain a smaller H_1-covering of S_{c}. In this section, we prove the minimality of the H_1-covering $\triangle_1$.

Theorem 11 *The equilateral triangle $\triangle_1$ is a minimal convex H_1-covering of S_{c}.*

Proof. Assume to the contrary that there is a proper subset T of $\triangle_1$ that is a convex H_1-covering of S_{c}. Since $\triangle_1$ is the convex hull of the corners of $\triangle_1$, T must be obtained by clipping some portions around some corners of $\triangle_1$. Since a vertical unit segment must be contained in T, no portion around the top corner of $\triangle_1$ can be cut off.

Let I_n be an isosceles triangle with perimeter 2 whose legs are of $1 - 1/3n$ each, base is parallel to x-axis, and apex is the top corner of I_n. Let I'_n be a copy of

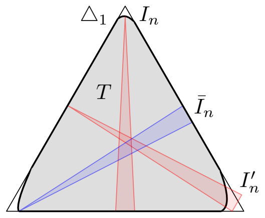


Figure 2: No proper subset T of $\triangle_1$ is a convex H_1-covering of S_c. The isosceles triangle I_n, a rotated copy I'_n of I_n by $\pi/3$, and a reflected copy $\bar{I}_n$ of I'_n along the x-axis.

I_n rotated by $\pi/3$. Observe that no translate of I'_n is contained in $\triangle_1$ for any positive integer n. See Figure 2. Since T is a proper subset of $\triangle_1$, no translate of I'_n is contained in T for any positive integer n. Thus, a reflected copy of I'_n along the x-axis is contained in T under translation for every n.

Let $\bar{I}_n$ be the reflection copy of I'_n such that $\bar{I}_n$ is contained in T. Since T is compact, there is a subsequence $\{\bar{I}_{n_i}\}$ that converges to a unit line segment with orientation $\pi/6$ contained in T. Observe that $\triangle_1$ contains a unit line segment with orientation $\pi/6$ only when the left endpoint of the segment lies at the bottom-left corner of $\triangle_1$. Thus, no portion around the bottom-left corner of $\triangle_1$ can be cut off. Similarly, no portion around the bottom-right corner of $\triangle_1$ can be cut off. Therefore, we conclude that that T is $\triangle_1$, contradicting that T is a proper subset of $\triangle_1$. Thus, $\triangle_1$ is a minimal H_1-covering of S_c. □

3.4 The smallest area triangle H_1-covering of S_c

Now we show that $\triangle_1$ has the smallest area among all triangle H_1-coverings of S_c. The following two lemmas describe geometric properties of a triangle that circumscribes a convex polygon P.

Lemma 12 ([16]) *If a triangle T has a local minimum in area among all triangles enclosing a convex polygon P, the midpoint of each side of T touches P.*

Following [19], we say that a side s of a triangle is *flush* with an edge e of P if $e \subseteq s$. Also, we say that a circumscribing triangle $\triangle$ is *P-anchored* if a side of $\triangle$ is flush with an edge of P and the other two sides of $\triangle$ touch vertices of P at their midpoints.

Lemma 13 (Lemma 1 of [19]) *For any P-anchored triangle $\triangle$, there always exists some P-anchored triangle $\triangle'$ such that $|\triangle| = |\triangle'|$, $\triangle$ and $\triangle'$ share one side, and at least two sides of $\triangle'$ are flush with edges of P.*

Recall that T_L given in Theorem 10 is an equilateral triangle of perimeter 2 such that it has a vertical side and its opposite corner lies to the right and T_R is a copy of T_L rotated by π.

Lemma 14 *Let Q be the convex hull of T_L and a translated copy of T_R. Then $|Q| \geq |T_L| + |T_R|$.*

The following lemma can be shown by Lemmas 12, 13, and 14.

Lemma 15 *The equilateral triangle $\triangle_1$ is the smallest triangle T-covering of T_L and T_R.*

Theorem 16 *The equilateral triangle $\triangle_1$ is the smallest-area triangle H_1-covering of S_c.*

Proof. Let $\triangle$ be a smallest-area triangle H_1-covering of S_c. Since $\triangle$ is H_1-covering, it is a covering of T_L and T_R under translation. By Lemma 15, $\triangle_1$ is the smallest-area triangle H_1-covering of S_c. □

A major open problem is whether $\triangle_1$ is a smallest-area H_1-covering of S_c. As Theorem 6 says, $\triangle_1$ is the smallest-area G_2-covering of S_c, and it is because $\triangle_1$ is the smallest-area G_2-covering of S_{seg}. However, as we have seen in Theorem 3, the smallest-area H_1-covering of S_{seg} is smaller, and has area 1/2. This is because a line segment (located so that its midpoint is at the origin) is stable under the rotation by π, but not stable under the reflection with respect to the x-axis unless it is horizontal or vertical.

4 Universal convex H_2-coverings of S_c

The dihedral group D_2 is generated by the reflection ρ with respect to the x-axis and the π-rotation g. Note that $g\rho$ is the reflection with respect to the y-axis.

The following result was given by Jung *et al.* [14]:

Theorem 17 *An equilateral triangle of height $\beta = \cos(\pi/12) \approx 0.966$ is a G_4-covering of S_c.*

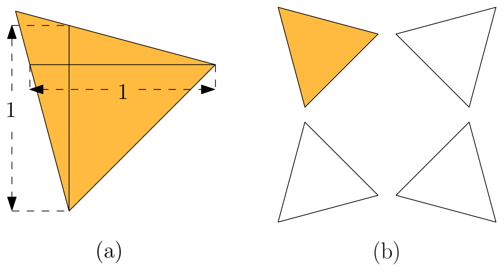


Figure 3: (a) An equilateral triangle $\triangle_\beta$ of height β containing horizontal and vertical unit line segments. (b) The triangle $\triangle_\beta$ and three triangles forming the D_2-orbit of $\triangle_\beta$.

Now, we consider $\triangle_\beta$ that is the equilateral triangle of height β located such that one of its sides has orientation $\pi/4$. Then, we have the following:

Theorem 18 *The equilateral triangle $\triangle_\beta$ is an H_2-covering of S_c. Moreover, it is minimal.*

Proof. Consider the D_2-orbit of $\triangle_\beta$. Then, as observed in Figure 3, they are exactly the same as the rotated copies of $\triangle_\beta$ with $k\pi/2$-rotations for $k = 0, 1, 2, 3$. Thus, it follows from Theorem 17 that $\triangle_\beta$ is an H_2-covering.

Consider the set A of unit length segments (regarded as degenerate closed curves of length 2) that are contained in $\triangle_\beta$. It is observed that $A' = A \setminus B$ has at most one element of each D_2-orbit of S_seg, where B is the set of six segments with orientations $k\pi/6$ for $k = 0, 1, \dots, 5$. Thus, each segment in A' must be contained in any H_2-covering $Q \subseteq \triangle_\beta$ under translation. By Theorem 1, there is a smallest-area T-covering of A' that is a triangle, and the algorithm given in [1] shows that $\triangle_\beta$ is the triangle. Thus, $Q = \triangle_\beta$, and hence $\triangle_\beta$ is minimal. □

We say that an object is *θ-orthogonal symmetric* if it has a pair of symmetry axes with orientations θ and $\theta + \pi/2$.

Corollary 19 *Any curve in S_c that is θ-orthogonal symmetric for either $\theta = 0, \pi/3$, or $2\pi/3$ can be contained in $\triangle_\beta$ by translation. In particular, any rectangle of perimeter 2 that has an edge with orientation either 0, $\pi/3$, or $2\pi/3$ can be contained in $\triangle_\beta$ by translation.*

Note that $\triangle_1$ is the smallest-area T-covering of the family of all rotated rectangles of perimeter 2 [14].

5 Universal convex H_3-coverings of S_c

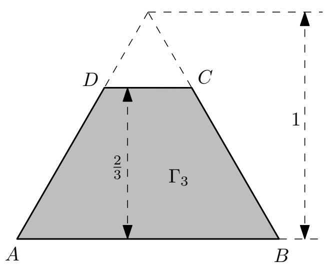


Figure 4: Construction of a convex H_3-covering Γ_3 of S_c. It is the trapezoid obtained from $\triangle_1$ after clipping a top part (an equilateral triangle) of height 1/3.

Let Γ_3 be the trapezoid obtained from $\triangle_1$ after clipping a top part (an equilateral triangle) of height 1/3. See Figure 4. Let A, B, C, D be the corners of Γ_3 in counterclockwise order, with A being the bottom-left corner. We show that Γ_3 is a convex H_3-covering of S_c.

A *slab* is the region bounded by two parallel lines in the plane, and its width is the distance between the lines. Let L_θ denote a slab of orientation θ, and let $w(L_\theta)$ be the width of L_θ.

Lemma 20 *For a closed curve β, let L_θ be the minimum-width slab of orientation θ that contains β. The length of β is at least $w(L_0) + w(L_{\pi/3}) + w(L_{2\pi/3})$.*

By Lemma 20, we have the following result.

Theorem 21 *The trapezoid Γ_3 is a convex H_3-covering of S_c.*

The area of Γ_3 is strictly smaller than that of the equilateral triangle of height $\beta = \cos(\pi/12)$.

6 Conclusion

This research is about how the mirror (i.e., reflection) effects on Moser's worm problems. Compared to the discrete rotation case given in [14], the positioning of the covering matters if we introduce the reflection, which requires delicate mathematical handling.

The research status of Moser's worm problems on S_c for T and TR remains rather static, and it is awkward to conjecture that the known small-area coverings are the optimal ones. In contrast to it, if we consider the affine dihedral groups such as H_1 and H_2, we can give clear conjectures that suitable equilateral triangles are the smallest-area coverings. The authors think they are mathematically clean and curious conjectures, and hope novel mathematical tools will be developed in the course of challenging to prove or disprove them.

Finally, although the H_k-coverings for $k \geq 4$ of S_seg have been classified, those for S_c have not been investigated, and it would be curious to find a unified approach to study them.

References

[1] H.-K. Ahn, S.-W. Bae, O. Cheong, J. Gudmundsson, T. Tokuyama, A. Vigneron, A Generalization of the Convex Kakeya Problem, *Algorithmica,* **70**:152-170, 2014.

[2] H.-K. Ahn, O. Cheong, Aligning Two Convex Figures to Minimize Area or Perimeter, *Algorithmica,* **62**:464-479, 2012.

[3] S. W. Bae, S. Cabello, O. Cheong, Y. Choi, F. Stehn, S. D. Yoon: The reverse Kakeya problem, *Advances in Geometry,* **21(1)**:75-84, 2021.

[4] J. Baez, K. Bagdasaryan, P. Gibbs, The Lebesgue universal covering problem, *Journal of Computational Geometry,* **6(1)**:288-299, 2015.

[5] A. Besicovitch, On Kakeya's problem and a similar one, *Mathematische Zeitschrift,* **27**:312-320, 1928.

[6] K. Bezdek, R. Connelly, Covering Curves by Translates of a Convex Set, *The American Mathematical Monthly,* **96(9)**:789-806, 1989.

[7] P. Brass, W. Moser, J. Pach, *Research Problems in Discrete Geometry*, Springer Verlag, 2005.

[8] H. T. Croft, K.J. Falconer, R.K. Guy, *Unsolved Problems in Geometry*, Springer-Verlag, 1991.

[9] Z. Dvir, On the size of Kakeya sets in finite fields. *J. Amer. Math. Soc.* **22**(4): 1093-1097, 2009.

[10] Z. Fürdi, J. W. Wetzel, Covers for Closed Curves of Length Two, *Periodia Mathematica Hungaria,* **63(1)**:1-17, 2011.

[11] P. Gibbs, An Upper Bound for Lebesgue's Covering Problem, arXiv:1810.10089, 2018.

[12] B. Grechuk, S. Som-am, A convex cover for closed unit curves has area at least 0.0975, *International Journal of Computational Geometry & Applications,* **30(2)**:121-139, 2020.

[13] B. Grechuk, S. Som-am, A convex cover for closed unit curves has area at least 0.1, *Discrete Optimization,* **38**, article 100608: 1-15, 2020.

[14] M. K. Jung, S. D. Yoon, H.-K. Ahn, T. Tokuyama, Universal convex covering problems under translation and discrete rotations, arXiv:2211.14807 [cs.CG] , 2022.

[15] S. Kakeya, Some problems on maximum and minimum regarding ovals, *Tohoku Science Report,* **6**:71-88, 1917.

[16] V. Klee, Facet-centroids and volume minimization, *Studia Scientiarum Mathematicarum Hungarica*, **21**:143-147 1986.

[17] L. Moser, Poorly formulated unsolved problems of combinatorial geometry, Mimeographed, 1966.

[18] W. O. J. Moser, Problems, problems, problems, *Discrete Applied Mathematics,* **31**:201-225, 1991.

[19] J. O'Rourke, A. Aggarwal, S. Maddila, M. Baldwin, An Optimal Algorithm for Finding Minimal Enclosing Triangles, *Journal of Algorithms,* **7**:258-269, 1986.

[20] R. Norwood, G. Poole, An improved upper bound for Leo Moser's worm problem, *Discrete Computational Geometry,* **29**:409-417, 2003.

[21] J. Pal, Ein Minimumproblem für Ovale, *Mathematische Annalen,* **83**:311-319, 1921.

[22] C. Panraksa, W. Wichiramala, Wetzel's sector covers unit arcs, *Periodica Mathematica Hungarica,* **82**:213-222, 2021.

[23] J. Park, O. Cheong, Smallest universal covers for families of triangles. *Computational Geometry,* **92**: 101686, 2021.

[24] T. Tao, From Rotating Needles to Stability of Waves: Emerging Connections between Combinatorics, Analysis, and PDE, *Notice of American Mathematical Society*, **48**(3): 294-303, 2001.

[25] J. E. Wetzel, Triangular covers of closed curves of constant length, *Elemente der Mathematik,* **25(4)**:78-82, 1970.

[26] J. E. Wetzel, On Moser's problem of accommodating closed curves in triangle, *Elemente der Mathematik,* **27(2)**:35-36, 1972.

[27] J. E. Wetzel, Sectorial covers for curves of constant length, *Canadian Mathematical Bulletin,* **16**:367-375, 1973.

[28] J. E. Wetzel, Fits and covers. *Mathematics Magazine,* **76**:349-363, 2003.

[29] W. Wichiramala, A smaller cover for closed unit curves, *Miskolc Mathematical Notes,* **19(1)**:691-698, 2018.

On the FPT Status of Monotone Convex Chain Cover

Qizheng He[*]

Abstract

Given a set of points P in the plane, the monotone convex cover number $\kappa(P)$ is the minimum number of x-monotone convex chains that can together cover P. We show the problem of deciding whether $\kappa(P) \leq k$ is NP-hard and does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly.

1 Introduction

The parameterized complexity of various geometric covering problems is a fundamental question related to clustering, and quite a bit of attention is attracted in recent years [9, 29, 2, 24, 25, 30, 7]. A typical formulation of the geometric covering problem is as follows: given a set of points P in $\mathbb{R}^d$ and a set of geometric objects S, find the smallest subset of objects from S that together cover all points in P.

We say a problem is Fixed-Parameter Tractable (F-PT), if there exists an algorithm with running time $f(k) \cdot |x|^{O(1)}$, where x is a string encoding a given problem instance, and k is the parameter.

For simple geometric objects, Langerman and Morin [28] presented an FPT algorithm framework to solve abstract geometric covering problems, with running time depending exponentially on the combinatorial dimension of the problem. This abstract setting models a number of concrete problems, e.g. covering points in $\mathbb{R}^2$ by k lines, or more generally covering points by k hyperplanes in $\mathbb{R}^d$ for constant dimension d. In this case, the combinatorial dimension equals to the geometric dimension d, and a simple bounded search tree and kernelization algorithm works [19, 28, 21, 38]. The problem can be solved in deterministic $O(k^{dk}n)$ time, or randomized $O(k^{d(k+1)} + 2^d k^{\lceil (d+1)/2 \rceil \lfloor (d+1)/2 \rfloor} n \log n)$ time.

The covering problem becomes more interesting for geometric objects with non-constant complexity, where less results were known in the literature. In this paper, we study the parameterized complexity of the problem of covering points in $\mathbb{R}^2$ by monotone convex chains.

The problem is defined as follows. Let $Q = (q_1, \ldots, q_m)$ be a sequence of points in the plane. The sequence Q is a *convex chain*, if $\forall\ 1 \leq i \leq m$, q_i is the i-th vertex of the convex hull $CH(Q)$ of Q. Furthermore, Q is an *x-monotone convex chain*, if $\forall\ 1 \leq i \leq m$, q_i is the i-th vertex of the lower hull $LH(Q)$ of Q (i.e., Q is a downward-convex point set, and the x-coordinates of the points q_i are monotonically increasing).

The *convex cover number* of a set of points P, denoted as $\kappa_c(P)$, is the minimum number of convex chains using only points in P that together cover all points in P [4]. The *monotone convex cover number* $\kappa(P)$ is defined similarly, further requiring that the convex chains would also be x-monotone. See Fig. 1 for an example.

Our goal is to determine the monotone convex cover number of a given point set P.

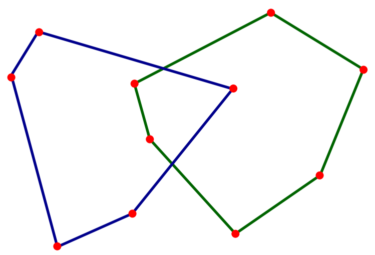
Problem 1 *(Monotone Convex Chain Cover) Given a set of points P in the plane and a parameter k, decide whether the monotone convex cover number $\kappa(P)$ of P is at most k.*

Arkin et al. proved that determining the convex cover number is NP-complete, and also presented an $O(\log n)$-approximation algorithm [4]. In this work, we similarly prove that computing the monotone convex chain cover number is also NP-complete.

A somewhat related problem is *monotone subsequence cover*, which asks for deciding whether a permutation π can be partitioned into k monotone subsequences. If we require all subsequences to be monotonically increasing, then the problem can be solved in polynomial time by finding the longest antichain, using Dilworth's theorem [13]. The problem becomes NP-hard when each subsequence is allowed to be either (monotonically) increasing or decreasing [37]. Heggernes et al. showed that this problem is FPT, by giving an algorithm that solves the problem in $2^{O(k^2 \log k)} n^{O(1)}$ time [23]. They reduced the monotone subsequence cover problem to the cochromatic number problem, and provided an FPT algorithm to compute the cochromatic number on perfect graphs.

The convex cover number is also related to other mathematical concepts, such as the convex partition number $\kappa_p(P)$, which asks for the minimum number of convex chains covering P, where the convex chains are required to be pairwise-disjoint convex hulls [4]. Computing the convex partition number exactly is also NP-complete, and an $O(\log n)$-approximation algorithm is known [4]. Another related concept is the reflexivity $\rho(P)$, which is the minimum number of reflex vertices (i.e., having interior angle $> \pi$) in a simple polygonalization of P [4, 1]. One more similar problem is

[*]Department of Computer Science, University of Illinois at Urbana-Champaign, qizheng6@illinois.edu


(a) P has convex cover number $\kappa_c(P) = 2$: P can be covered by two convex chains as shown, and easy to verify that $\kappa_c(P) > 1$, because $CH(P) \subsetneq P$.

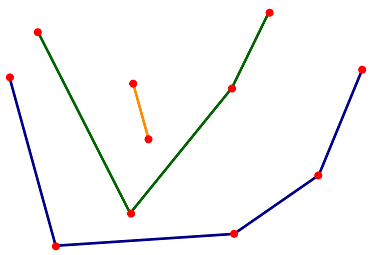

(b) P has monotone convex cover number $\kappa(P) = 3$.

Figure 1: The convex cover number and monotone convex cover number of a point set P.

to compute a planar subdivision of the input point set P, where each bounded interior face is a convex polygon, and minimizing the number of such convex polygon faces. Polynomial-time constant factor approximation algorithm exists [26], but the best known exact algorithm runs in $n^{O(k)}$ time for deciding whether this number is at most k [18]. If we parameterize by the number of inner points k' in P, this problem is known to be FPT [20, 34].

Despite the extensive studies for related problems, there are only few previous works focusing on the complexity of monotone convex chain cover. A natural conjecture is that the monotone convex chain cover problem we study here is harder than the monotone subsequence cover problem, due to the additional convexity constraint (constraints on triples instead of pairs of points). One particular open question asked by Eppstein is whether convex chain cover is FPT [15, Open Problem 11.16], and the same question can also be asked for monotone convex chain cover. Here we provide a conditional hardness result showing that monotone convex chain cover does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly, taking a step towards resolving the parameterized complexity of the problem.

Worst-case convex cover number. There are previous results on bounding the convex cover number in the worst case. Erdős and Szekeres proved that any point set with size n contains a convex subset with size $\Omega(\log n)$ [16, 17]. This is related to the famous happy ending problem [16], which had been recently (nearly) resolved by Suk [35]. As a consequence, Urabe showed that $\kappa_c(n) = \Theta(\frac{n}{\log n})$ [36], where $\kappa_c(n)$ denotes the worst-case convex cover number of a set of n points. The hidden constant in the Θ-notation is also of interest to some researchers, see e.g. [33].

For the monotone convex cover number, the trivial $\Theta(n)$ bound is tight: consider the input points forming a concave set as shown in Fig. 2, in this case the monotone convex cover number and the convex cover number are far apart. In the random case and the grid case, better bounds are known [22, 12].

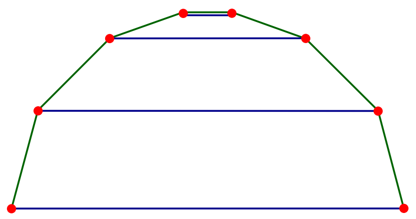


Figure 2: A concave point set P with even size n has convex cover number $\kappa_c(P) = 1$ (green chain) and monotone convex cover number $\kappa(P) = n/2$ (blue chains).

Approximation algorithms. It is not hard to get a polynomial time approximation algorithm for computing the convex cover number, with an $O(\log n)$-approximation factor [4]. The idea is to use the greedy algorithm for general set cover, where in each iteration we greedily choose the convex chain with the largest number of points in the remaining set of points. Note that we can compute the largest convex chain in polynomial time, by dynamic programming [32]. This algorithm readily leads to an $O(\log n)$-approximation for the monotone convex cover number. An open question is whether getting constant factor approximation is possible for either of the problems.

XP algorithm. A simple observation is the convex chain cover problem can be solved in $n^{O(k)}$ time, using dynamic programming [15, Section 11.5]. The same algorithm works for monotone convex chain cover, and for completeness we briefly redescribe it here. Suppose we want to decide whether $\kappa(P) \le k$. In each state of dynamic programming, we maintain k x-monotone convex chains (may be empty), and keep track of the last two points of each monotone convex chain. Sort the input points according to their x-coordinates in increasing order, and attach them one by one to the tails of the convex chains, enumerating each combination. We can verify whether the current point can be added to a particular convex chain while maintaining convexity, using its relative position with the stored last two points of the chain. There are $n^{O(k)}$ states, and computing the value of each state takes polynomial time, thus this problem is in XP. This is still the fastest algorithm known to the best of our knowledge.

2 Our Results

2.1 NP-hardness for Monotone Convex Chain Cover

We first show that the monotone convex chain cover problem is NP-hard, which serves as a building block for proving nonexistence of polynomial kernels later.

It is known that computing the convex cover number $\kappa_c(P)$ for a given point set P is NP-hard, as shown by Arkin et al. [4] (which turns out to be inspired by the hardness proof for Angular Metric TSP by Aggarwal et al. [3]). Since convex chain cover and monotone convex chain cover are closely related, by slightly modifying their proof, we are able to prove a similar hardness result for monotone convex chain cover.

Theorem 1 *Given a set of points P in the plane and an integer k, it is NP-hard to decide whether the monotone convex cover number $\kappa(P)$ of a point set P is at most k.*

We first sketch the argument of Arkin et al.'s NP-hardness proof, then highlight the modifications we made for proving NP-hardness for monotone convex chain cover.

Proof. Arkin et al.'s NP-hardness proof for convex cover number is via a reduction from the 1-in-3 SAT problem, i.e. deciding whether there exists a satisfying assignment of the given formula, such that exactly one literal in each clause is true. In their reduction, for each clause they created three columns, where each column corresponds to a variable, and for each variable they created two rows, corresponding to the true and false assignments. Then they added "pivot" points to encode the variable-clause incidence information. Finally, they added "staple" gadgets to the rows and columns each containing sufficiently large number of points which are in convex position, one staple for each variable and two staples for each clause, and the aim is to use those gadgets as convex chains to cover all the "pivot" points. The 1-in-3 SAT instance I is satisfiable iff $\kappa_c(P_I) \le n + 2m$, where n is the number of variables and m is the number of clauses.

Our modifications for monotone convex chain cover are as follows. Here one can only use x-monotone convex chains to cover the points, therefore one needs to replace each "staple" gadget with a pair of x-monotone convex chains (the red curves illustrated in Fig. 3), each is slightly perturbed from a straight line segment and contains $\Omega(n^4)$ points, which is sufficiently large to ensure each pair of such monotone convex chains will be selected as a whole. Each "staple" can be linked as a single x-monotone convex chain that is able to cover all the pivot points in one row or one column, but cannot cover any other pivot point. We carefully set the positions and the slopes of the monotone convex chains to ensure each pair of convex chains that is created will be contained in a single x-monotone convex chain in the optimal solution. One need to also slightly rotate the whole point set S_I counterclockwise by a negligibly small angle, in order to ensure x-monotonicity of the (nearly vertical) chains that we want to appear in the optimal solution. The rest of the correctness proof follows from Arkin et al.'s work.

An example of our modified reduction construction is shown in Fig. 3. □

Remark. Intuitively, the monotone convex chain cover problem is hard to solve, since it is known that covering points by lines is NP-hard [31, 14] and even APX-hard [8, 27], and monotone convex chains are more complicated geometric objects than lines. However, we are not aware of a direct reduction between these two problems.

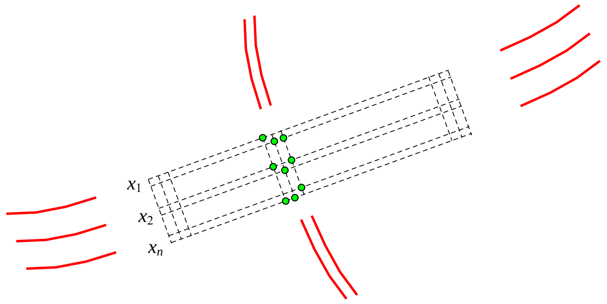


Figure 3: The point set P_I we create for a 1-in-3 SAT instance I for reducing to the monotone convex chain cover problem. Each red curve illustrates a "staple" gadget, which is in fact a sufficiently long x-monotone convex chain. The pivot points for the clause $(x_1 \vee \overline{x_i} \vee x_n)$ are shown in green.

2.2 No Polynomial Kernel

Next, we show that the monotone convex chain cover problem does not have a polynomial sized kernel, unless NP $\subseteq$ coNP/poly, which is believed to be unlikely, as it will imply that the polynomial hierarchy collapses [39, 10].

Kernelization. A *kernel* for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm $\mathcal{A}$ that, given an instance (I, k) of Q, returns an equivalent instance (I', k') of Q

in polynomial time such that $(I, k) \in Q$ iff $(I', k') \in Q$. The size of the kernel $|I'| + k'$ is required to be upper bounded by $f(k)$ for some computable function f [11]. We say the problem Q has a *polynomial kernel*, if there exists a polynomial such function f.

It is well-known that a parameterized problem is FPT iff it admits a kernelization algorithm [11].

Composing problem instances. The key observation is that given t point sets $P_1, \ldots, P_t$, we can create a new point set $\hat{P}$ such that its monotone convex cover number $\kappa(\hat{P}) \leq k$ iff $\kappa(P_i) \leq k$ for all $1 \leq i \leq t$.

The detailed construction is as follows. Note that convexity is preserved under affine transformations, in particular translation, rotation and scaling. The idea is to apply an affine transformation on each point set P_i to get a new point set $\hat{P}_i$, and let $\hat{P}$ be the disjoint union of all $\hat{P}_i$'s. We first scale each point set P_i to ensure that $\hat{P}_i$ is contained in an axis-aligned bounding box which is wide and narrow. In particular, after applying appropriate horizontal and vertical scaling, we can w.l.o.g. assume the width of the bounding box is 1, and the absolute value of the slope between each pair of points in $\hat{P}_i$ is bounded by ε, where $\varepsilon = \varepsilon(t)$ is a sufficiently small value depending only on t. As a consequence, the height of the bounding box is bounded by 2ε. Next, we rotate the bounding box of $\hat{P}_i$ by $\frac{i}{t}$ radians counterclockwise, and finally translate the bounding box to let its lower left corner lie on coordinate $(\sum_{j=0}^{2i-3} \cos \frac{j}{2t}, \sum_{j=0}^{2i-3} \sin \frac{j}{2t})$, specifically $(0, 0)$ for $\hat{P}_1$. See Fig. 4 for an example.

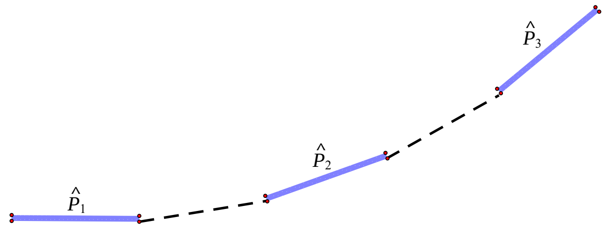


Figure 4: The point set $\hat{P}$ created from composing P_1, P_2 and P_3. Each transformed point set $\hat{P}_i$ is contained in a bounding box shown in blue.

The above construction ensures that an x-monotone convex chain in P_i is still an x-monotone convex chain in $\hat{P}$ after applying the corresponding affine transformation. Moreover, for any pair $i < j$, we can concatenate any x-monotone convex chain in $\hat{P}_i$ with any other x-monotone convex chain in $\hat{P}_j$, to get a single x-monotone convex chain in $\hat{P}$ after the transformation. Therefore $\kappa(\hat{P}_i \cup \hat{P}_j) \leq \max\{\kappa(\hat{P}_i), \kappa(\hat{P}_j)\}$ by greedily concatenating the chains in $\hat{P}_i$ with the chains in $\hat{P}_j$. On the other hand, clearly $\kappa(\hat{P}_i) \leq \kappa(\hat{P}_i \cup \hat{P}_j)$ as $\hat{P}_i \subseteq \hat{P}_i \cup \hat{P}_j$, and similarly $\kappa(\hat{P}_j) \leq \kappa(\hat{P}_i \cup \hat{P}_j)$. Thus, $\kappa(\hat{P}_i \cup \hat{P}_j) = \max\{\kappa(\hat{P}_i), \kappa(\hat{P}_j)\}$.

As a result, $\hat{P}$ can be covered by at most k x-monotone convex chains iff each part P_i can be covered by at most k x-monotone convex chains.

AND-composition. The above seemingly simple construction implies we can compose multiple instances of monotone convex chain cover into a single instance, where the answer for the combined instance equals to the Boolean AND of the answers for those instances. The next step is to use the framework of Bodlaender et al. [5, 6] to prove that the monotone convex chain cover problem does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly. To begin with, we introduce the concept of AND-composition.

Definition 2 (AND-composition [5]) *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. An AND-composition algorithm for L is an algorithm that takes as input a sequence $((x_1, k), \ldots, (x_t, k))$ where $(x_i, k) \in \Sigma^* \times \mathbb{N}^+$ $\forall\ 1 \leq i \leq t$, runs in time polynomial in $\sum_{i=1}^{t} |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}^+$ such that:*
(a) $(y, k') \in L$ iff $(x_i, k) \in L\ \forall\ 1 \leq i \leq t$, and
(b) $k' = \mathrm{poly}(k)$.

We claim that our above construction gives an AND-composition algorithm for the parameterized language of the monotone convex chain cover problem. One can encode a point set P containing n points using poly(n) bits, because one only need the information about the rank of the x-coordinates of the points, and the convexity of each triple of points. Namely, the input can be encoded using $O(n^3)$ bits. It is easy to verify that the definitions of AND-composition are met.

Bodlaender et al. [5] showed that if an AND-compositional parameterized language L has a polynomial kernel, and if its unparameterized version $\tilde{L}$ is NP-complete, then NP $\subseteq$ coNP/poly. Combining our AND-composition construction in this section with the NP-hardness result in Sec. 2.1, we obtain the following theorem:

Theorem 2 *The monotone convex chain cover problem parameterized by k does not admit a polynomial kernel, unless* NP $\subseteq$ coNP/poly.

Open problems. We conclude with a number of open problems to explore for future work:

- Our results exclude the existence of a polynomial kernel for monotone convex chain cover, unless NP $\subseteq$ coNP/poly. But whether this problem is FPT remains open.

- Since monotone convex chain cover and convex chain cover seem to be closely related, it would be interesting to see whether we can prove similar kernelization hardness results for convex chain cover. Our current technique does not obviously generalize, because in our construction for AND-composition, x-monotonicity is crucial for combining multiple monotone convex chains into a single one. Another direction is to directly relate the hardness between monotone convex/concave chain cover and convex chain cover.

- The current best polynomial time approximation algorithms for convex chain cover and monotone convex chain cover only achieve $O(\log n)$-approximation. It is not known whether polynomial time constant factor approximation algorithms exist for these two problems, or whether the problems are APX-hard.

Acknowledgement. We thank Sariel Har-Peled for helpful discussions and, in particular, for bringing the convex chain cover problem to our attention.

References

[1] Eyal Ackerman, Oswin Aichholzer, and Balázs Keszegh. Improved upper bounds on the reflexivity of point sets. *Comput. Geom.*, 42(3):241–249, 2009.

[2] Pankaj K. Agarwal and Cecilia Magdalena Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

[3] Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The angular-metric traveling salesman problem. *SIAM Journal on Computing*, 29(3):697–711, 2000.

[4] Esther M Arkin, Joseph SB Mitchell, Sándor P Fekete, Ferran Hurtado, Marc Noy, Vera Sacristán, and Saurabh Sethia. On the reflexivity of point sets. *Discrete and Computational Geometry*, pages 139–156, 2003.

[5] Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Danny Hermelin. On problems without polynomial kernels. In *International Colloquium on Automata, Languages, and Programming (I-CALP)*, pages 563–574, 2008.

[6] Hans L Bodlaender, Bart MP Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

[7] Karl Bringmann, Sándor Kisfaludi-Bak, Michal Pilipczuk, and Erik Jan van Leeuwen. On geometric set cover for orthants. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA)*, pages 26:1–26:18, 2019.

[8] Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding lines and 2-link polygons is APX-hard. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG)*, pages 45–48, 2001.

[9] Sergio Cabello, Panos Giannopoulos, Christian Knauer, Dániel Marx, and Günter Rote. Geometric clustering: Fixed-parameter tractability and lower bounds with respect to the dimension. *ACM Trans. Algorithms*, 7(4):43:1–43:27, 2011. Preliminary version in SODA'08.

[10] Jin-yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Inf. Comput.*, 198(1):1–23, 2005.

[11] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. 2015.

[12] Ketan Dalal. Counting the onion. *Random Structures & Algorithms*, 24(2):155–165, 2004.

[13] Robert P Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[14] Adrian Dumitrescu and Minghui Jiang. On the approximability of covering points by lines and related problems. *Comput. Geom.*, 48(9):703–717, 2015.

[15] David Eppstein. *Forbidden configurations in discrete geometry*. Cambridge University Press, 2018.

[16] Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio mathematica*, 2:463–470, 1935.

[17] Paul Erdős and George Szekeres. On some extremum problems in elementary geometry. In *Annales Univ. Sci. Budapest*, pages 3–4, 1960.

[18] Thomas Fevens, Henk Meijer, and David Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1-2):95–107, 2001.

[19] Panos Giannopoulos, Christian Knauer, and Sue Whitesides. Parameterized complexity of geometric problems. *The Computer Journal*, 51(3):372–384, 2008.

[20] Magdalene Grantson and Christos Levcopoulos. A fixed parameter algorithm for the minimum number convex partition problem. In *Discrete and Computational Geometry, Japanese Conference (JCDCG), Revised Selected Papers*, volume 3742 of *Lecture Notes in Computer Science*, pages 83–94, 2004.

[21] Magdalene Grantson and Christos Levcopoulos. Covering a set of points with a minimum number of lines. In Tiziana Calamoneri, Irene Finocchi, and Giuseppe F. Italiano, editors, *Algorithms and Complexity, 6th Italian Conference (CIAC)*, volume 3998 of *Lecture Notes in Computer Science*, pages 6–17, 2006.

[22] Sariel Har-Peled and Bernard Lidicky. Peeling the grid. *SIAM Journal on Discrete Mathematics*, 27(2):650–655, 2013.

[23] Pinar Heggernes, Dieter Kratsch, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing via iterative localization. *Information and Computation*, 231:109–116, 2013.

[24] R. Z. Hwang, R. C. Chang, and Richard C. T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica*, 9(4):398–423, 1993.

[25] R. Z. Hwang, Richard C. T. Lee, and R. C. Chang. The slab dividing approach to solve the Euclidean p-center problem. *Algorithmica*, 9(1):1–22, 1993.

[26] Christian Knauer and Andreas Spillner. Approximation algorithms for the minimum convex partition problem. In Lars Arge and Rusins Freivalds, editors, *10th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 4059 of *Lecture Notes in Computer Science*, pages 232–241, 2006.

[27] V. S. Anil Kumar, Sunil Arya, and H. Ramesh. Hardness of set cover with intersection 1. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1853 of *Lecture Notes in Computer Science*, pages 624–635. Springer, 2000.

[28] Stefan Langerman and Pat Morin. Covering things with things. In *European Symposium on Algorithms (ESA)*, pages 662–674, 2002.

[29] Dániel Marx. Efficient approximation schemes for geometric problems? In Gerth Stølting Brodal and Stefano Leonardi, editors, *Proceedings of the 13th Annual European Symposium (ESA)*, volume 3669 of *Lecture Notes in Computer Science*, pages 448–459, 2005.

[30] Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA)*, pages 865–877, 2015.

[31] Nimrod Megiddo and Arie Tamir. On the complexity of locating linear facilities in the plane. *Oper. Res. Lett.*, 1(5):194–197, 1982.

[32] Joseph SB Mitchell, Günter Rote, Gopalakrishnan Sundaram, and Gerhard J Woeginger. Counting convex polygons in planar point sets. *Inf. Process. Lett.*, 56(1):45–49, 1995.

[33] J Pach. Discrete and computational geometry, 19. *Special issue dedicated to Paul Erdös*, 1998.

[34] Andreas Spillner. Optimal convex partitions of point sets with few inner points. In *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG)*, pages 39–42, 2005.

[35] Andrew Suk. On the Erdős-Szekeres convex polygon problem. *Journal of the American Mathematical Society*, 30(4):1047–1053, 2017.

[36] Masatsugu Urabe. On a partition of point sets into convex polygons. In *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG)*, 1997.

[37] Klaus W. Wagner. Monotonic coverings of finite sets. *J. Inf. Process. Cybern.*, 20(12):633–639, 1984.

[38] Jianxin Wang, Wenjun Li, and Jianer Chen. A parameterized algorithm for the hyperplane-cover problem. *Theor. Comput. Sci.*, 411(44-46):4005–4009, 2010.

[39] Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983.

On Density Extrema for ℓ_1-Balls in 2D and 3D Integer Space

Nilanjana G. Basu[*] Subhashis Majumder[*] Partha Bhowmick[†]

Abstract

Digital balls are made of integer points and defined in a particular finite-dimensional metric space. By their very definition, they indeed have a drastic difference from the real-space balls because their elements are countable. The countability offers the scope for their unique characterization and related applications in discrete-geometric computation in various domains such as computer vision and combinatorial image analysis. Density is one of the unique characteristics of digital balls, and since this measure varies with the position and the size of a ball in any metric space, a natural inquisition lies with its extremum values. This paper presents some notable results on these extrema for ℓ_1-balls in 2D and 3D space. Further possible investigations related to this are also mentioned at the end.

1 Introduction

Characterization of integer points for various computational purposes is one of the prevalent problems in computer graphics, computer vision, and image analysis. It helps us understand various facts and figures that are intuitively not apparent. Opposed to points in the real space, points in the integer space are countable, equipped with certain classes of neighborhood relations that are different from those in the real space, and can be used to constitute specialized topological spaces with appropriate metrics. Counting and density measure of integer points are two related concepts that are needed to characterize specific metric-defined balls or simple geometric shapes. One such classic example is Pick's Theorem for triangles on the 2D plane, given that the vertices are all integer points. The scenario, however, becomes quite complex when the vertices are not bound to be integer points, especially for polygons (or polyhedra) with four or more vertices. The proof technique for one class of polygons may not be handy for another class, and hence exclusive proofs are needed for them. In this paper, we present some novel findings on minimum and maximum densities of ℓ_1-balls in 2D and 3D integer space.

[*]Department of Computer Science & Engineering, Heritage Institute of Technology, Kolkata, India. {nilanjanag.basu,subhashis.majumder}@heritageit.edu

[†]Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur, India. pb@cse.iitkgp.ac.in

Table 1: Summary of results. (ε is a real number tending to 0. The entries marked by $\star$ are not yet found by us.)

			max density		min density	
	Center	λ	ρ	λ	ρ	λ
2-diamond	$\mathbb{Z}^2$	$\mathbb{Z}^+$	$\frac{5}{2}$	2	1	∞
	$\mathbb{Z}^2$	$\mathbb{R}^+$	$\frac{5}{2}$	2	$\frac{1}{2}$	$2-\lvert\varepsilon\rvert$
	$\mathbb{R}^2$	$\mathbb{Z}^+$	4	1	$\star$	$\star$
	$\mathbb{R}^2$	$\mathbb{R}^+$	4	1	$\frac{4}{9}$	$3-\lvert\varepsilon\rvert$
3-diamond	$\mathbb{Z}^3$	$\mathbb{Z}^+$	$\frac{21}{4}$	2	1	∞
	$\mathbb{Z}^3$	$\mathbb{R}^+$	$\frac{21}{4}$	2	$\frac{21}{32}$	$4-\lvert\varepsilon\rvert$
	$\mathbb{R}^3$	$\mathbb{Z}^+$	12	1	$\star$	$\star$
	$\mathbb{R}^3$	$\mathbb{R}^+$	12	1	$\frac{3}{8}$	$4-\lvert\varepsilon\rvert$

1.1 Related work

Density, as a measure, provides a notion of the relative concentration of points within a given shape or region [13]. Hence, it finds numerous applications in different branches of physical science; some of the interesting ones can be found in [6, 7, 10, 11, 16, 17, 18] and in the references therein.

Quite a handful of work has been done related to digital discs and digital balls defined on square or non-square grid [8, 5, 14]. There are also some research works on different algorithmic techniques and analyses related to their constructions [1, 2, 4, 15]. Regarding 2-dimensional digital ℓ_2-balls (i.e., Euclidean discs), certain interesting results on minimum and maximum densities can be seen in a recent paper [3].

1.2 Our work

An ℓ_1-ball can be conceived as a diamond in 2D and a regular octahedron in 3D. Its density is defined as the number of integer points per unit of its area or volume. (Mathematical definitions follow shortly in §1.3.) We have characterized the density functions of ℓ_1-balls in 2D and 3D with all possible specifications and have thereby deduced the definite values of maximum and minimum densities. Interestingly, these extremum values are found to depend on whether the diagonal-lengths or/and the centers are real- or integer-valued. The main results are summarized in Table 1.2. We denote by $\mathbb{R}$ the set of real numbers, by $\mathbb{Z}$ the set of integers, and by $\mathbb{Z}^+$ the set of positive integers.

1.3 Conventions and Terminologies

We consider uniform rectilinear grids in 2D and 3D. The points under consideration are essentially grid points, i.e., points of intersection among the grid lines or the grid planes. Consequently, the set of grid points in 2D (3D) are in bijection with the points in $\mathbb{Z}^2$ ($\mathbb{Z}^3$). Hence, the points are assumed to have integer coordinates and referred to as *integer points* for brevity. For definiteness, these integer points are also called *pixels* in 2D and *voxels* in 3D.[1] An ℓ_1-ball is essentially a diamond in 2D, i.e., a square with two axes-parallel diagonals. Similarly, in 3D, it is a regular octahedron with three axes-parallel diagonals. For simplicity, we call them 2-*diamond* and 3-*diamond*, respectively; and we specify either of them by the center and the length of a diagonal. Further, by the term "real diagonal" or "integer diagonal", we mean a diagonal of real or integer length, respectively. Examples of 2-diamonds and 3-diamonds are shown in Figure 1.

For $n = 2$ and 3, we denote by $\mathrm{D}_\lambda^{(n)}$ an n-diamond having a diagonal of length λ. We do not include its center in this notation to keep it simple, but mention it whenever needed.

Considering the center as the origin in the local coordinate system in $\mathbb{R}^n$, and denoting the ℓ_1 norm of a point p by $\|p\|_1$, an n-diamond is thus defined as

$$\mathrm{D}_\lambda^{(n)} := \left\{p \in \mathbb{R}^n : 2 \cdot \|p\|_1 \leq \lambda\right\}$$

and its *boundary* is given by

$$\mathrm{bdy}\left(\mathrm{D}_\lambda^{(n)}\right) := \left\{p \in \mathbb{R}^n : 2 \cdot \|p\|_1 = \lambda\right\}.$$

The set of integer points contained in $\mathrm{D}_\lambda^{(n)}$ is referred to as a *digital n-diamond*, and it is given by

$$\mathbb{D}_\lambda^{(n)} := \mathrm{D}_\lambda^{(n)} \cap \mathbb{Z}^n = \left\{p \in \mathbb{Z}^n : 2 \cdot \|p\|_1 \leq \lambda\right\}.$$

The cardinality of $\mathbb{D}_\lambda^{(n)}$ is denoted by $|\mathbb{D}_\lambda^{(n)}|$, and the area (if $n = 2$) or volume (if $n = 3$) of $\mathrm{D}_\lambda^{(n)}$ by $\mathrm{vol}(\mathrm{D}_\lambda^{(n)})$. The density of integer points in $\mathbb{D}_\lambda^{(n)}$ is given by

$$\rho_\lambda^{(n)} = \frac{\left|\mathbb{D}_\lambda^{(n)}\right|}{\mathrm{vol}\left(\mathrm{D}_\lambda^{(n)}\right)}.$$

2 Maximum density

For characterization of maximum-density diamonds in 2D and 3D, we consider those containing more than one integer point, because containment of just one integer point is trivial and degenerates to the limiting case of infinite density.

[1] In the literature of computer graphics and digital image processing, integer points in 2D are referred to as *pixels*, and those in 3D are referred to *voxels*. This is mathematically well-founded because 2D/3D integer points are isomorphic to pixels/voxels [12].

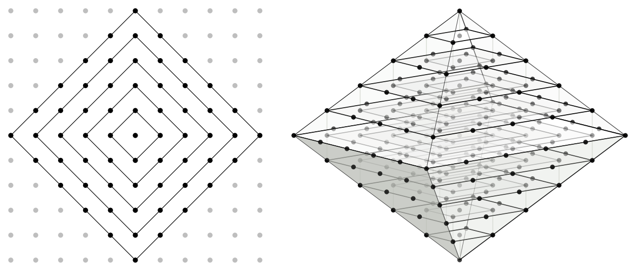


Figure 1: 2-diamonds (left) and 3-diamonds (right) with integer center and diagonal-length $\lambda \in \{0, 2, 4, \ldots, 10\}$.

In order to deduce the maximum density for 2-diamonds and 3-diamonds with real-valued diagonals, we now make an important observation that comes to use in narrowing down our attention to a countable collection of diamonds. For $n = 2, 3$, let $\mathrm{D}_\lambda^{(n)}$ be a diamond that does not contain any integer point on its boundary. Then we can always shrink it uniformly, keeping its center fixed, without missing any of its integer points, until some integer point $p \in \mathbb{D}_\lambda^{(n)}$ touches its boundary. Let $\mathrm{D}_\mu^{(n)}$ be the diamond after shrinking, such that p lies on its boundary. Clearly, $\mathrm{D}_\mu^{(n)} \subsetneq \mathrm{D}_\lambda^{(n)}$ and $\mathbb{D}_\mu^{(n)} = \mathbb{D}_\lambda^{(n)}$, which implies that $\rho_\mu^{(n)} > \rho_\lambda^{(n)}$, whence the following observation.

Observation 1 *Any 2-diamond or any 3-diamond of maximum density must contain an integer point on its boundary.*

2.1 Integer center

Observe that a 2-diamond or a 3-diamond having an integer center and a diagonal of length λ doesn't contain any integer point on its boundary if λ is an odd integer. So, by Observation 1, it cannot not have the maximum density. In fact, such a diamond $\mathrm{D}_\lambda^{(n)}$ always contains within itself a higher-density diamond $\mathrm{D}_{\lambda-1}^{(n)}$ whose diagonal-length (i.e., $\lambda - 1$) is an even integer. Thus, we make the following observation.

Observation 2 *A 2-diamond or a 3-diamond centered at an integer point can have the maximum density only if its diagonal-length is an even integer.*

Based on the above observation, we deduce the maximum densities for integer-centered 2-diamonds and 3-diamonds, as stated in the following two theorems.

Theorem 1 (2-diamonds: integer center) *In the collection of all 2-diamonds with integer centers, those with maximum density have diagonals of length 2 and density $\frac{5}{2}$.*

Proof. Consider a 2-diamond with integer center and of diagonal-length $\lambda = 2k$, where k is a positive integer, as shown in Figure 1. By a simple counting, we get

$$|\mathbb{D}^{(2)}_{2k}| = (2k+1) + 2\sum_{i=1}^{k}(2i-1) = 2k^2 + 2k + 1. \quad (1)$$

Hence,

$$\rho^{(2)}_{2k} = \frac{|\mathbb{D}^{(2)}_{2k}|}{\mathrm{vol}(\mathrm{D}^{(2)}_{2k})} = \frac{2k^2+2k+1}{2k^2} = 1 + \frac{1}{k} + \frac{1}{2k^2}. \quad (2)$$

Since $\arg\max_k \left\{\rho^{(2)}_{2k}\right\} = 1$ is unique, the maximum density occurs if and only if $\lambda = 2$, whence the result. □

Theorem 1 can be extended to derive a similar result for 3-diamonds having integer center, as stated next.

Theorem 2 (3-diamonds: integer center) *In the collection of all 3-diamonds with integer centers, those with maximum density have diagonals of length 2 and density $\frac{21}{4}$.*

Proof. By Observation 2, a 3-diamond $\mathrm{D}^{(3)}_{\lambda}$ has maximum density only if $\lambda = 2k$, where k is a positive integer. The corresponding digital 3-diamond, i.e., $\mathbb{D}^{(3)}_{\lambda}$, is given by the union of the central 2-diamond, i.e., $\mathbb{D}^{(2)}_{\lambda}$, and its upper and lower 2-diamonds of diagonal-lengths $2(k-1), 2(k-2), \ldots, 0$. Thus,

$$\begin{aligned}\left|\mathbb{D}^{(3)}_{2k}\right| &= \left|\mathbb{D}^{(2)}_{2k}\right| + 2\sum_{i=0}^{k-1}\left|\mathbb{D}^{(2)}_{2i}\right| \\ &= \tfrac{4}{3}k^3 + 2k^2 + \tfrac{8}{3}k + 1.\end{aligned} \quad (3)$$

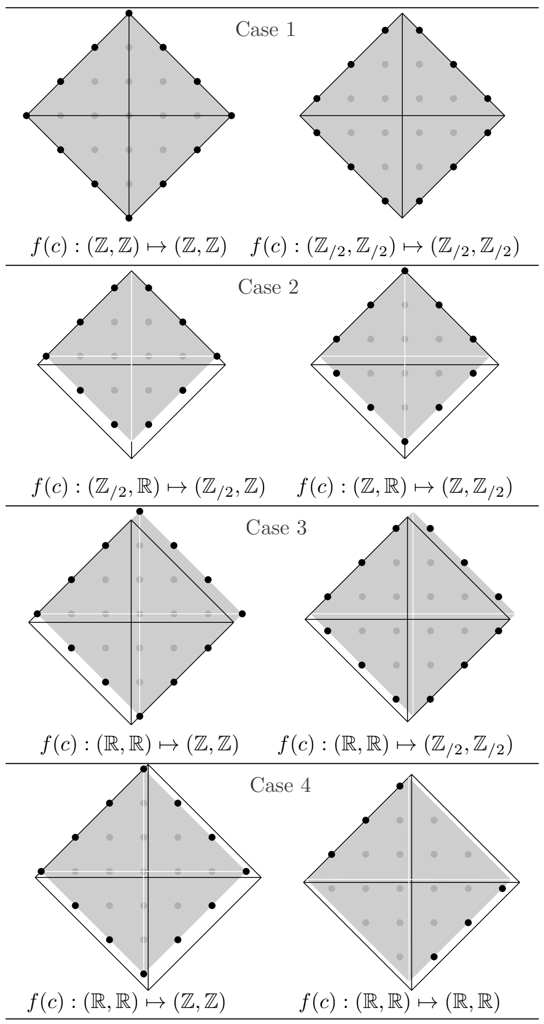
As $\mathrm{vol}\left(\mathrm{D}^{(3)}_{2k}\right) = \frac{\sqrt{2}}{3}(k\sqrt{2})^3 = \frac{4k^3}{3}$, we get

$$\rho^{(3)}_{2k} = \frac{|\mathbb{D}^{(3)}_{2k}|}{\mathrm{vol}(\mathrm{D}^{(3)}_{2k})} = 1 + \frac{3}{2k} + \frac{2}{k^2} + \frac{3}{4k^3}. \quad (4)$$

The inference now is similar to the proof of Theorem 1: $\arg\max_k \left\{\rho^{(2)}_{2k}\right\} = 1$ is unique, which implies that the maximum density is for a unique value of k, which gives $\lambda = 2$, and the corresponding density is $\frac{21}{4}$. □

2.2 Real center

By Observation 1, to achieve maximum density, any 2-diamond $\mathrm{D}^{(2)}_{\lambda}$ must have at least one integer point on its boundary. We characterize its center c based on this. Let f be a function that maps c to a nearest point $f(c)$ in 2D for which there exists a 2-diamond $\mathrm{D}^{(2)}_{\mu}$ centered at $f(c)$ such that $\left|\mathbb{D}^{(2)}_{\mu}\right| = \left|\mathbb{D}^{(2)}_{\lambda}\right|$ and $\mu \le \lambda$. Let $c = (c_x, c_y)$ and $f(c) = (f_x(c), f_y(c))$. W.l.o.g.,



Case 1
$f(c) : (\mathbb{Z},\mathbb{Z}) \mapsto (\mathbb{Z},\mathbb{Z})$
$f(c) : (\mathbb{Z}_{/2},\mathbb{Z}_{/2}) \mapsto (\mathbb{Z}_{/2},\mathbb{Z}_{/2})$
Case 2
$f(c) : (\mathbb{Z}_{/2},\mathbb{R}) \mapsto (\mathbb{Z}_{/2},\mathbb{Z})$
$f(c) : (\mathbb{Z},\mathbb{R}) \mapsto (\mathbb{Z},\mathbb{Z}_{/2})$
Case 3
$f(c) : (\mathbb{R},\mathbb{R}) \mapsto (\mathbb{Z},\mathbb{Z})$
$f(c) : (\mathbb{R},\mathbb{R}) \mapsto (\mathbb{Z}_{/2},\mathbb{Z}_{/2})$
Case 4
$f(c) : (\mathbb{R},\mathbb{R}) \mapsto (\mathbb{Z},\mathbb{Z})$
$f(c) : (\mathbb{R},\mathbb{R}) \mapsto (\mathbb{R},\mathbb{R})$

Figure 2: Result of applying f on the center c of a 2-diamond—four exhaustive cases, each with a couple of sub-cases. $\mathrm{D}^{(2)}_{\lambda}$ are shown with black diagonals, and $\mathrm{D}^{(2)}_{\mu}$ shown in gray with white diagonals.

we take $f_x(c) = c_x + \delta_x$ and $f_y(c) = c_y + \delta_y$, where $(\delta_x, \delta_y) \in \left[-\frac{1}{2}, \frac{1}{2}\right]^2$. Clearly, $\mathrm{D}^{(2)}_{\lambda}$ cannot have maximum density if $\mu < \lambda$. We analyze with four exhaustive cases given below, which are also illustrated in Figure 2. We denote by $\mathbb{Z}_{/2}$ the set of half-integers, i.e., $\mathbb{Z}_{/2} = \left\{a + \frac{1}{2} : a \in \mathbb{Z}\right\}$.

[1] All four sides of $\mathrm{D}^{(2)}_{\lambda}$ contain integer points: $c \in \mathbb{Z}^2 \uplus \mathbb{Z}^2_{/2} \implies f(c) = c \implies \lambda \in \mathbb{Z}$.

[2] Only two adjacent sides of $\mathrm{D}^{(2)}_{\lambda}$ contain integer points: $c \in ((\mathbb{R} \times \mathbb{Z}_{/2}) \cup (\mathbb{Z}_{/2} \times \mathbb{R})) \cup ((\mathbb{R} \times \mathbb{Z}) \cup (\mathbb{Z} \times \mathbb{R})) \implies f(c) \in \mathbb{Z} \times \mathbb{Z}_{/2} \implies \mu < \lambda$. So, $\mathrm{D}^{(2)}_{\lambda}$ cannot have maximum density.

[3] Only two opposite sides of $\mathrm{D}^{(2)}_{\lambda}$ contain integer points: A careful observation shows that $f(c) \in \mathbb{Z}^2 \uplus \mathbb{Z}^2_{/2}$ and $\mu = \lambda$, which implies λ is an integer.

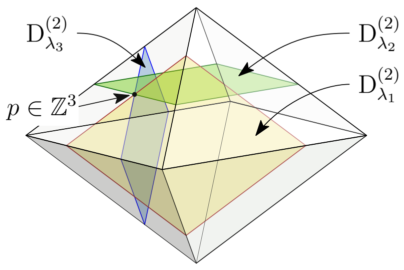

$\mathrm{D}^{(2)}_{\lambda_3}$
$\mathrm{D}^{(2)}_{\lambda_2}$
$\mathrm{D}^{(2)}_{\lambda_1}$
$p \in \mathbb{Z}^3$

Figure 3: Three 2-diamonds in $\mathrm{D}^{(3)}_\lambda$ such that the integer point p on the boundary of $\mathrm{D}^{(3)}_\lambda$ lies on their boundaries as well.

[4] Only one side of $\mathrm{D}^{(2)}_\lambda$ contain integer points: We get $\mu < \lambda$ by shrinking, which implies $\mathrm{D}^{(2)}_\lambda$ cannot have maximum density.

We extend the above analysis to 3-diamonds with real centers. By Observation 1, any 3-diamond $\mathrm{D}^{(3)}_\lambda$ has maximum density only if at least one of its eight faces contains an integer point p. As shown in Figure 3, this point p will lie on the boundaries of exactly three 2-diamonds, namely $\mathrm{D}^{(2)}_{\lambda_i}$, where $i = 1, 2, 3$. Each of them is parallel to one of the principal planes. Consider any of them—w.l.o.g., say it is $\mathrm{D}^{(2)}_{\lambda_1}$. We refer back to Figure 2. If its boundary admits Case 1 or Case 3, then $\lambda_1 \in \mathbb{Z} \implies \lambda \in \mathbb{Z}$; otherwise, $\mathrm{D}^{(2)}_{\lambda_1}$ can be shrunk to a smaller 2-diamond $\mathrm{D}^{(2)}_{\mu_1}$ so that $\mu_1 < \lambda_1$ and $\left|\mathbb{D}^{(2)}_{\mu_1}\right| = \left|\mathbb{D}^{(2)}_{\lambda_1}\right|$. Each other 2-diamond in $\mathrm{D}^{(3)}_\lambda$, which is parallel to $\mathrm{D}^{(2)}_{\lambda_1}$ and of the form $\mathrm{D}^{(2)}_{\lambda_1+k}$, where $k \in \mathbb{Z}$, admits this shrinking. So, $\mathrm{D}^{(3)}_\lambda$ cannot have maximum density. This gives the following observation.

Observation 3 *Any 2-diamond or any 3-diamond with a real center can have the maximum density only if the length of its diagonal is an integer, and then its respective center will be in* $(\mathbb{Z} \uplus \mathbb{Z}_{/2})^2$ *or in* $(\mathbb{Z} \uplus \mathbb{Z}_{/2})^3$.

By Observation 3, if the center c of a 2-diamond or of a 3-diamond is in $\mathbb{Z}^2$ or in $\mathbb{Z}^3$, then the results are already known (Theorems 1 and 2). So, we now find the results if c is in $(\mathbb{Z} \uplus \mathbb{Z}_{/2})^2 \smallsetminus \mathbb{Z}^2$ or in $(\mathbb{Z} \uplus \mathbb{Z}_{/2})^3 \smallsetminus \mathbb{Z}^3$.

Theorem 3 (2-diamonds: real center) *In the collection of all 2-diamonds with centers in* $\mathbb{R}^2 \smallsetminus \mathbb{Z}^2$ *and with integer diagonals, the ones with maximum density have centers in* $(\mathbb{Z} \uplus \mathbb{Z}_{/2})^2 \smallsetminus \mathbb{Z}^2$*, diagonals of unit length, and density* 4.

Proof. We prove in two parts—one for odd and one for an even value of λ (Figure 4). Let c be the center of a 2-diamond $\mathrm{D}^{(2)}_\lambda$.

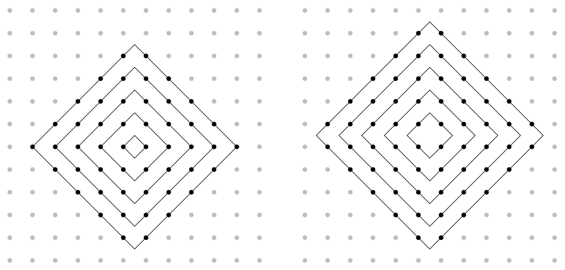


Figure 4: Two possible cases of $\mathrm{D}^{(2)}_\lambda$ with the centers in $(\mathbb{Z} \uplus \mathbb{Z}_{/2})^2 \smallsetminus \mathbb{Z}^2$: odd λ (left) and even λ (right).

Part i. Let $\lambda = 2k - 1$, where $k \in \mathbb{Z}^+$. By Observation 3, $c \in (\mathbb{Z} \times \mathbb{Z}_{/2}) \uplus (\mathbb{Z}_{/2} \times \mathbb{Z})$. We have

$$\left|\mathbb{D}^{(2)}_{2k-1}\right| = 2k + 2\sum_{i=1}^{k-1} 2i = 2k^2 \tag{5}$$

$$\implies \rho^{(2)}_{2k-1} = \frac{\left|\mathbb{D}^{(2)}_{2k-1}\right|}{\mathrm{vol}(\mathbb{D}^{(2)}_{2k-1})} = \frac{2k^2}{\frac{1}{2}(2k-1)^2} = \left(\frac{1}{1-\frac{1}{2k}}\right)^2. \tag{6}$$

As $\rho^{(2)}_{2k-1}$ decreases monotonically with k, the maximum density occurs at $k = 1$, and its value is 4.

Part ii. Let $\lambda = 2k$, where $k \in \mathbb{Z}^+$. By Observation 3, $c \in \mathbb{Z}_{/2} \times \mathbb{Z}_{/2}$. We have

$$\left|\mathbb{D}^{(2)}_{2k}\right| = 2\sum_{i=1}^{k} 2i = 2k(k+1) \tag{7}$$

$$\implies \rho^{(2)}_{2k} = \frac{\left|\mathbb{D}^{(2)}_{2k}\right|}{\mathrm{vol}(\mathbb{D}^{(2)}_{2k})} = \frac{2k(k+1)}{\frac{1}{2}(2k)^2} = 1 + \frac{1}{k}. \tag{8}$$

As in Part i, $\arg\max_k \rho^{(2)}_{2k} = 1$, and so $\max \rho^{(2)}_{2k} = 2$. Combining Part i and Part ii, we get the result. ☐

The above theorem can be extended to obtain the results for 3-diamonds, as stated in the following theorem.

Theorem 4 (3-diamonds: real center) *In the collection of all 3-diamonds with centers in* $\mathbb{R}^3 \smallsetminus \mathbb{Z}^3$ *and with integer diagonals, the ones with maximum density have centers in* $(\mathbb{Z} \uplus \mathbb{Z}_{/2})^3 \smallsetminus \mathbb{Z}^3$*, diagonals of unit length, and density* 12.

Proof. We prove in three parts depending on the location of the center c.

Part i. $c \in (\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}_{/2}) \uplus (\mathbb{Z} \times \mathbb{Z}_{/2} \times \mathbb{Z}) \uplus (\mathbb{Z}_{/2} \times \mathbb{Z} \times \mathbb{Z})$, $\lambda = 2k - 1$, where $k \in \mathbb{Z}^+$.

$$\left|\mathbb{D}^{(3)}_{2k-1}\right| = 2k^2 + 2\sum_{i=1}^{k-1} 2i^2 \left.\right\} \text{by union of the central digital 2-diamond } \mathbb{D}^{(2)}_{2k-1} \text{ and the digital 2-diamonds on its either sides}$$
$$= \frac{4}{3}k^3 + \frac{2}{3}k. \tag{9}$$

As $\mathrm{vol}\left(\mathbb{D}^{(3)}_{2k-1}\right) = \frac{2^3\left(\frac{2k-1}{2}\right)^3}{3!} = \frac{(2k-1)^3}{6}$, we get

$$\rho^{(3)}_{2k-1} = \frac{\frac{4}{3}k^3 + \frac{2}{3}k}{\frac{(2k-1)^3}{6}} = \frac{8k^3+4k}{(2k-1)^3} = \frac{1+\frac{1}{2k^2}}{\left(1-\frac{1}{2k}\right)^3} \tag{10}$$

which decreases monotonically with k and maximizes at $k = 1$, with the value 12.

Part ii. $c \in (\mathbb{Z} \times \mathbb{Z}_{/2} \times \mathbb{Z}_{/2}) \uplus (\mathbb{Z}_{/2} \times \mathbb{Z} \times \mathbb{Z}_{/2}) \uplus (\mathbb{Z}_{/2} \times \mathbb{Z}_{/2} \times \mathbb{Z})$, $\lambda = 2k$, where $k \in \mathbb{Z}^+$.

$$\left|\mathbb{D}^{(3)}_{2k}\right| = 2k(k+1) + 2\sum_{i=1}^{k-1} 2i(i+1)$$
$$= \frac{4}{3}k^3 + 2k^2 + \frac{2}{3}k. \tag{11}$$

As $\mathrm{vol}\left(\mathbb{D}^{(3)}_{2k}\right) = \frac{2^3k^3}{3!} = \frac{4}{3}k^3$, we get

$$\rho^{(3)}_{2k} = \frac{\frac{4}{3}k^3 + 2k^2 + \frac{2}{3}k}{\frac{4}{3}k^3} = 1 + \frac{3}{2k} + \frac{1}{2k^2} \tag{12}$$

which decreases monotonically with k and maximizes at $k = 1$, with the value 3.

Part iii. $c \in \mathbb{Z}_{/2} \times \mathbb{Z}_{/2} \times \mathbb{Z}_{/2}$, $\lambda = 2k+1$, $k \in \mathbb{Z}^+$.

$$\left|\mathbb{D}^{(3)}_{2k+1}\right| = 2\sum_{i=1}^{k} 2i(i+1) \left.\right\} \text{no central digital 2-diamond; digital 2-diamonds are only on two sides of } \mathrm{D}^{(3)}_{2k+1}$$
$$= \frac{4}{3}k(k+1)(k+2). \tag{13}$$

As $\mathrm{vol}\left(\mathbb{D}^{(3)}_{2k+1}\right) = \frac{2^3\left(\frac{2k+1}{2}\right)^3}{3!} = \frac{(2k+1)^3}{6}$, we get

$$\rho^{(3)}_{2k+1} = \left(\frac{2k}{2k+1}\right)\left(\frac{2k+2}{2k+1}\right)\left(\frac{2k+4}{2k+1}\right) \tag{14}$$

which also decreases monotonically with k (derivative always negative) and maximizes at $k = 1$, with the value $\frac{16}{9}$. Combining the three parts, we get the result. □

3 Minimum density

We present here some results on the characteristics of minimum-density diamonds in 2D and 3D. A diamond without any integer point has zero density and is disregarded from our consideration. Our deductions are based on a generic fact that a diamond with minimal density will be maximal in size with no integer point on its boundary. In particular, we use the following observation.

Observation 4 *For a fixed center c, if* $\mathrm{D}^{(n)}_{\lambda_0}$ *is a diamond with center c and with* $\mathrm{bdy}\left(\mathrm{D}^{(n)}_{\lambda_0}\right) \cap \mathbb{Z}^n \neq \varnothing$*, then the minimal-density diamond that has center c and contains all and only the integer points of* $\mathrm{D}^{(n)}_{\lambda_0}$*, will have the maximal-length diagonal, and that length is*

$$\arg\sup_{\lambda}\left\{\mathrm{vol}\left(\mathrm{D}^{(n)}_{\lambda}\right) : \left(\mathrm{bdy}\left(\mathrm{D}^{(n)}_{\lambda}\right) \cap \mathbb{Z}^n = \varnothing\right)\right\}.$$

3.1 Integer center and integer diagonal

As shown in §2, for integer center and integer diagonal, a diamond always has even-length diagonals, i.e., it is of the form $\mathbb{D}^{(n)}_{2k}$, where $n = 2, 3$ and $k \in \mathbb{Z}^+$. From (2) and (4), it can be noticed that the density function of $\mathbb{D}^{(n)}_{2k}$ decreases monotonically with k. Further, $\inf_{k\in\mathbb{Z}^+}\left\{\rho^{(n)}_{2k}\right\} = 1$. Hence, a minimum-density diamond has infinite diagonal-length and density 1 in both 2D and 3D.

3.2 Integer center and real diagonal

Unlike §3.1, the result for 2-diamond and that for 3-diamond are different in this setting. We present them exclusively in the following two theorems.

Theorem 5 (2-diamond: integer-real) *In the collection of all 2-diamonds with integer center and real diagonal, the diagonal-length of the ones with minimum density is the largest real number less than* 2.

Proof. We refer back to (2). For a maximal-size real diamond containing a single pixel, the density is

$$\lim_{\varepsilon\to 0}\left(\rho^{(2)}_{2-|\varepsilon|}\right) = \lim_{\varepsilon\to 0}\left(\frac{1}{\mathrm{vol}\left(\mathrm{D}^{(2)}_{2-|\varepsilon|}\right)}\right) = \frac{1}{2}.$$

For $k \geq 2$, we have

$$\lim_{\varepsilon\to 0}\left(\rho^{(2)}_{2k-|\varepsilon|}\right) = \frac{2k^2-2k+1}{2k^2} = 1 - \frac{1}{k}\left(1 - \frac{1}{2k}\right) > \frac{1}{2},$$

whence the result. □

Theorem 6 (3-diamond: integer-real) *In the collection of all* 3*-diamonds with integer center and real diagonal, the diagonal-length of the ones with minimum density is the largest real number less than* 4.

Proof. We refer back to (4). Let λ denote the diagonal-length. There is a maximal-size 3-diamond containing 7 voxels having λ just less than 4. The supremum of its density is $\frac{7}{\frac{4}{3}\cdot 2^3} = \frac{21}{32}$. For $\lambda < 2$, there exists only a unique case of maximal-size diamond containing a single voxel, and its density is

$$\lim_{\varepsilon\to 0} \rho^{(3)}_{2-|\varepsilon|} = \frac{1}{\frac{4}{3}\cdot 1^3} = \frac{3}{4} > \frac{21}{32}. \tag{15}$$

For $k \geq 1$, we have

$$\begin{aligned}\lim_{\varepsilon\to 0} \rho^{(3)}_{2k-|\varepsilon|} &= \frac{\left|\mathbb{D}^{(3)}_{2k-1}\right|}{\mathrm{vol}\left(\mathrm{D}^{(3)}_{2k-|\varepsilon|}\right)} \\ &= \frac{\frac{4}{3}(k-1)^3 + 2(k-1)^2 + \frac{8}{3}(k-1) + 1}{\frac{4}{3}k^3} \\ &= 1 - \frac{3}{2k} + \frac{2}{k^2} - \frac{3}{4k^3} \geq \frac{21}{32}. \end{aligned} \tag{16}$$

Note that $k \geq 1 \implies (k-1)^2(11k-26) + (k+2) \geq 0 \implies 11k^3 - 48k^2 + 64k - 24 \geq 0$, which gives $\frac{21}{32}$ as the lower bound in (16). Now, from (15) and (16), the result follows. □

3.3 Real center and real diagonal

As per Observation 4, we need to consider only the following cases. We denote the center of $\mathrm{D}^{(2)}_{\lambda}$ by c.

1. $c \in \mathbb{Z}^2$: tackled in §3.2.
2. $c \in (\mathbb{Z} \times \mathbb{Z}_{/2}) \uplus (\mathbb{Z}_{/2} \times \mathbb{Z})$: λ will be just less than an odd integer.
3. $c \in \mathbb{Z}_{/2} \times \mathbb{Z}_{/2}$: λ will be just less than an even integer.

We have the following result for the last two cases.

Theorem 7 (2-diamond: real-real) *In the collection of all 2-diamonds with real center and real diagonal, the diagonal-length of the ones with minimum density is the largest real number less than* 3.

Proof. By Observation 4, the minimum density correspond to some λ just less than an integer. We consider the odd and the even lengths separately. Let $k \in \mathbb{Z}^+$. For the odd case, $\lambda = 2k-1$. For $k \leq 2$, there exists a unique case of maximal diamond containing two pixels, and its density is $\lim_{\varepsilon\to 0} \rho^{(2)}_{3-|\varepsilon|} = \frac{2}{\frac{1}{2}\cdot 3^2} = \frac{4}{9}$. Now, $k > 2 \implies (k-\frac{4}{5})(k-2) \geq 0 \implies \lim_{\varepsilon\to 0} \rho^{(2)}_{2k-1-|\varepsilon|} = \frac{4(k-1)^2}{(2k-1)^2} \leq \frac{4}{9}$. For the even case, $\lambda = 2k$. We have $\lim_{\varepsilon\to 0} \rho^{(2)}_{4-|\varepsilon|} = \frac{4}{\frac{1}{2}\cdot 4^2} = \frac{1}{2}$, which is less than $\lim_{\varepsilon\to 0} \rho^{(2)}_{2k-|\varepsilon|} = \frac{k-1}{k} = 1 - \frac{1}{k}$ for all $k > 2$, whence the proof. □

The above result for 2-diamonds and the one in Theorem 4 can be combined to get the following theorem. As the proof is similar to those done in Theorem 7 and Theorem 4, we give here a brief sketch of the proof.

Theorem 8 (3-diamond: real-real) *In the collection of all 3-diamonds with real center and real diagonal, the diagonal-length of the ones with minimum density is the largest real number less than* 4.

Proof. As per Observation 4 and as explained in Theorem 4, we have three possible cases for the location of center c, as each of its three coordinates can be in $\mathbb{Z}_{/2}$ or $\mathbb{Z}$. After analyzing these cases, we find that the minimum density occurs when c is in $\mathbb{Z} \times \mathbb{Z}_{/2} \times \mathbb{Z}_{/2}$ or an equivalent 3D space. We get

$$\rho^{(3)}_{2k} = \frac{\frac{4}{3}(k-1)^3 + 2(k-1)^2 + \frac{2}{3}(k-1)}{\frac{4}{3}k^3} = 1 - \frac{3}{2k} + \frac{1}{2k^2}$$

which is 0 if $k = 1$ (notice that such a 3-diamond exists). For $k = 2$, its value is $\frac{3}{8}$, and thereon it increases monotonically. □

4 Conclusion and future work

In this work, we have introduced an interesting problem of defining and characterizing density measure for ℓ_1-balls in 2D and 3D integer space. We have found the solutions for most of the possible configurations related to real and integer specifications except the minimum density cases for arbitrary centers and integer diagonals. Note that the diamonds (both 2D and 3D) with integer diagonals become minimal whenever the centers are aligned with a grid line along any direction, i.e., they have pixels on their boundary. So, these diamonds are natural candidates for maximum density. The maximal diamonds on the other hand are the natural candidates to compete for minimum density and with integer diagonals they stand at a disadvantage for being maximal under any circumstances. So to find the minimum density diamonds one has to consider a number of non-maximal diamonds also and that makes the search space somewhat different from the other cases and hence may require different techniques for their solutions. Such a difficulty was observed for ℓ_2-balls also and those cases are still open [3].

For ℓ_∞-balls, the solutions are rather easier, as we observed [9]. For ℓ_2-balls, we have already cited [3] in §1.1. However, for other metrics ℓ_p with $p > 2$, the characterization seems more elusive and likely to involve deeper mathematical analysis, even in 2D and 3D.

We have used uniform rectilinear grid to define the digital ℓ_1-balls. As a result, the discrete set of points are mapped to integer space. For other classes of grids and lattices, the problem is also definable and demands similar characterization of density measure. This is particularly interesting when we consider a vector space and the balls therein.

Last but not the least, the work reported in this paper will have stronger impact if it can be extended to higher dimensions, as higher-dimensional norms and metrics are inherently related to information theory and signal processing.

References

[1] E. Andres and T. Roussillon. Analytical description of digital circles. In I. Debled-Rennesson, E. Domenjoud, B. Kerautret, and P. Even, editors, *Discrete Geometry for Computer Imagery - 16th IAPR International Conference, DGCI 2011, Nancy, France, April 6-8, 2011. Proceedings*, volume 6607 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 2011.

[2] T. Barrera, A. Hast, and E. Bengtsson. A chronological and mathematical overview of digital circle generation algorithms introducing efficient 4- and 8-connected circles. *International Journal of Computer Mathematics*, 93(8):1241–1253, 2016.

[3] N. G. Basu, P. Bhowmick, and S. Majumder. On density extrema for digital discs. In R. P. Barneva, V. E. Brimkov, and G. Nordo, editors, *Combinatorial Image Analysis - 21st International Workshop, IWCIA 2022, Messina, Italy, July 13-15, 2022, Proceedings*, volume 13348 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 2022.

[4] P. Bhowmick and B. B. Bhattacharya. Number-theoretic interpretation and construction of a digital circle. *Discret. Appl. Math.*, 156(12):2381–2399, 2008.

[5] P. Bhowmick and B. B. Bhattacharya. Real polygonal covers of digital discs – some theories and experiments. *Fundam. Informaticae*, 91(3-4):487–505, 2009.

[6] G. R. Blake and K. Hartge. Particle density. *Methods of soil analysis: Part 1 physical and mineralogical methods*, 5:377–382, 1986.

[7] V. Cheng. Understanding density and high density. In *Designing high-density cities*, pages 37–51. Routledge, 2009.

[8] S. Fisk. Separating point sets by circles, and the recognition of digital disks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(4):554–556, 1986.

[9] N. G. Basu, P. Bhowmick, and S. Majumder. On density of grid points in l_∞-balls. In *Proceedings of 3rd International Conference on Mathematical Modeling and Computational Science(ICMMCS 2023), February, 24-25, 2023, Tamilnadu, India.* Springer Singapore, 2023.

[10] G. Hammarhjelm. The density and minimal gap of visible points in some planar quasicrystals. *Discret. Math.*, 345(12):113074, 2022.

[11] P. J. Hasnip, K. Refson, M. I. J. Probert, J. R. Yates, S. J. Clark, and C. J. Pickard. Density functional theory in the solid state. In *Phil. Trans. R. Soc. A.* Royal Society, 2014.

[12] R. Klette and A. Rosenfeld. *Digital Geometry: Geometric Methods for Digital Picture Analysis.* Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Francisco, 2004.

[13] S. Majumder and B. B. Bhattacharya. On the density and discrepancy of a 2d point set with applications to thermal analysis of vlsi chips. *Inf. Process. Lett.*, 107(5):177–182, 2008.

[14] B. Nagy. Number of words characterizing digital balls on the triangular tiling. In N. Normand, J. V. Guédon, and F. Autrusseau, editors, *Discrete Geometry for Computer Imagery - 19th IAPR International Conference, DGCI 2016, Nantes, France, April 18-20, 2016. Proceedings*, volume 9647 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2016.

[15] S. Pham. Digital circles with non-lattice point centers. *Vis. Comput.*, 9(1):1–24, 1992.

[16] J. Rakun, D. Stajnko, and D. Zazula. Plant size estimation based on the construction of high-density corresponding points using image registration. *Comput. Electron. Agric.*, 157:288–304, 2019.

[17] S. Wang, Q. Li, C. Zhao, X. Zhu, H. Yuan, and T. Dai. Extreme clustering - A clustering method via density extreme points. *Inf. Sci.*, 542:24–39, 2021.

[18] Y. Wang, D. Wang, W. Pang, C. Miao, A. Tan, and Y. Zhou. A systematic density-based clustering method using anchor points. *Neurocomputing*, 400:352–370, 2020.

List of Authors